AD-A118 822 ASSESSMENT SYSTEMS CORP ST PAUL MN F/G 9/2
DEVELOPMENT OF A MICROCOMPUTER-BASED ADAPTIVE TESTING SYSTEM. P--ETC(U)
JUN 82 C D VALE, C ALBING, L FOOTE-LENNOX N00014-82-C-0132
UNCLASSIFIED ONR-ASC-82-01 NL

1 of 2
A
118822

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE — READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER Research Report ONR-ASC-82-01	2. GOVT ACCESSION NO. AD-A118 822	3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*
Development of a Microcomputer-Based Adaptive Testing System
Phase I--Specification of Requirements and Preliminary Design

5. TYPE OF REPORT & PERIOD COVERED
Final Report: 1 January 1982 to 30 June 1982

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
C. David Vale
Carl Albing
Lisa Foote-Lennox
Thom Foote-Lennox

8. CONTRACT OR GRANT NUMBER(s)
N00014-82-C-0132

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Assessment Systems Corporation
2395 University Avenue, Suite 306
St. Paul, MN 55114

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
W.U.: NR 150-481

11. CONTROLLING OFFICE NAME AND ADDRESS
Office of Naval Research
Department of the Navy
Arlington, VA 22217

12. REPORT DATE
30 June 1982

13. NUMBER OF PAGES
175

14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*

15. SECURITY CLASS. *(of this report)*
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

adaptive testing	latent trait theory
computerized testing	microcomputer testing system
tailored testing	adaptive testing computer system design
item response theory	microcomputer evaluation

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Specifications were developed for a microcomputer-based adaptive testing system based on reviews of item types in current use and a survey of the needs and desires of individuals engaged in adaptive testing research and other related activities. From these specifications, a general system configuration was developed and the design of a general-purpose software package was initiated. This design included subsystems for test construction, item banking, test administration, test reporting, and test analysis. Various

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20.

microcomputer systems were considered in search of hardware on which to implement the design; of these, three eight-bit microcomputer systems were selected for future consideration. A draft of a user's manual, reviewed by experts, was developed for the proposed system and served as a preliminary design document. In summary, this research indicated that a definite need exists for a microcomputer-based adaptive testing system and that the proposed system would meet that need.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

PREFACE

This report summarizes the first phase in a project conducted to develop an inexpensive microcomputer-based adaptive testing system. Its focus was to determine the needs and desires of potential users of the system and to design a system to meet those needs. Later phases, if funded, will result in the development and commercial availability of such a system.

Many individuals participated in this project, some through interviews, some by completing questionnaires, and some by extensive assistance. The authors would like to collectively express their appreciation to all of these individuals. In addition, the authors would like to extend their special thanks to James McBride, Malcolm Ree, and David Weiss for critically reviewing the draft User's Manual and to Stephen Prestwood, Kathleen Gialluca, David Weiss, and Sandra Whelan for advice and editing of this report.

TABLE OF CONTENTS

PRECEDING PAGE

I. SPECIFICATION OF THE SYSTEM REQUIREMENTS

The interactive administration of psychological tests by computer has been a popular idea for more than a decade. Originally, the main impetus for the concept was to let a computer take over some of the mundane work of psychologists or psychometrists. As the concept grew, however, and psychometric theory caught up with the concept, it became obvious that the computer could not only perform the work of the psychometrist, it could actually do it more effectively. The computer, with statistical formulae and appropriate data, could quickly perform efficient score computations. Combining psychometric concepts with the interactive administration of tests by computer resulted in computerized adaptive testing (CAT).

A basic problem became apparent when the amount of programming required for adaptive testing was considered. Even though the computer was theoretically capable of replacing the psychometrist, in practice this feat was accomplished only in specific areas. It is easy to program a computer to administer a single, fixed-length, conventional multiple-choice test. It is more difficult to program multiple tests, variable-length tests, and adaptive tests. Tests with non-objective responses are still more difficult to implement.

A few computerized testing systems have been developed. For the most part, they have been small, special-purpose systems or large research systems developed by a single organization for an internal purpose. Thus, they are not general purpose or portable. Without a portable, psychometrically capable system, entering the computerized testing field has involved considerable developmental work to make a computerized test operational. The need for a capable, portable CAT system is obvious.

The obstacle in the development of such a system has been that a capable CAT system is usually somewhat dependent on the computer hardware; transfer from one machine to another is therefore difficult. Since large computers are expensive, they usually cannot be dedicated exclusively to testing. A research organization's testing system thus has had to run on the systems available. Rarely did many testing organizations have compatible equipment.

The availability of microcomputer systems and components has changed this, however. Capable microcomputers are sufficiently inexpensive that they can be dedicated solely to testing. It is now possible to design a testing system around a single computer and still have a system that is cost effective for testing.

A portable, dedicated CAT system is feasible, and it is reasonable to expect that the demand for such a system could be great. To meet this demand, such a system should be able to administer tests of most formats and strategies (both adaptive and conventional) that could benefit from computer administration. A comprehensive list and description of potential system requirements was compiled as a first step in the design of a system meeting these needs.

The analysis of system requirements will be presented in four parts. Part 1 will present a comprehensive list of item types currently in existence. Part 2 will consider testing strategies for selecting and ordering items; adaptive testing models will be reviewed in detail. Part 3 will describe a survey of system requirements as seen by potential system users. Finally, Part 4 will synthesize the requirements to provide basic specifications for the system.

A Survey of Item Types

Test items can be grouped into four general categories: (1) knowledge, (2) cognitive process, (3) perceptual-motor, and (4) non-cognitive. Each of these areas will be discussed in turn, including descriptions of a variety of item types within each area.

Knowledge Items

A test of knowledge attempts to assess how much information an examinee brings to the testing situation. Knowledge tests are probably the most appropriate for written administration because knowledge can be assessed by simply asking the examinee whether s/he can recall or recognize certain information. Knowledge tests are the most common type encountered, and consist exclusively of items in the form of questions. The format in which these questions are asked can vary widely (cf. Wesman, 1971).

The most common question format uses dichotomously scored multiple-choice items. These items present a question followed by two to five alternative answers, only one of which is correct. The examinee's task is to choose the correct alternative. The majority of these items can be presented in textual format and require no pictures or special characters that are not available on a standard CRT terminal. Some do require a picture or a drawing for the stem, however (e.g., the Mechanical Comprehension items of the Armed Services Vocational Aptitude Battery, or ASVAB). A few require pictures or drawings for both the stem and the alternatives. Responses are usually limited to a single number or letter within a small range.

A variation of the simple multiple-choice item is the answer-until-correct item (Hanna, 1975). The initial presentation of this type of item is identical to that of the simple multiple-choice item. If the examinee answers incorrectly, however, the item is presented again with the previously attempted alternatives eliminated or disabled. In computerized presentation, this may be done by eliminating the chosen alternatives from the CRT screen. The item is presented until the examinee answers correctly or until all of the incorrect alternatives are exhausted.

The confidence-weighted multiple-choice item format (Shuford, Albert, & Massengill, 1966) is another method of extracting more information from the simple multiple-choice item. In this format, the examinee answers the multiple-choice item and is then asked to indicate his/her level of confidence that the choice is correct. Administratively, this is a matter of following each knowledge question with a confidence question by which the examinee indicates his/her degree of confidence that the answer was correct.

Another variation, probabilistic-response items (de Finneti, 1965), incorporates the examinee's confidence into the initial item response. Rather than choosing a specific alternative, the examinee assigns probabilities to each alternative in accordance with his/her confidence that each is correct. Administratively, these items require a simple method for allowing the examinee to assign the probabilities to the appropriate alternatives and a method for editing and correcting the probabilities when they do not add up to 1.0.

Although the multiple-choice response has been the method most widely implemented for knowledge items, the free-response item has seen some limited use in computerized testing. For item types requiring a numerical response, such as an arithmetic item, the free-response mode is only slightly more difficult to implement than the multiple-choice mode. Simple textual-response items have also been successfully implemented on a computer (e.g., Vale & Weiss, 1977; Vale, 1978). The problem with free-response items comes not in accepting the response, but in processing the response into a form that can be scored by the computer. In the example by Vale (1978), the free responses were categorized for further statistical processing. More complex (e.g., essay) free-response modes have not yet been successfully computerized, although some computer-assisted-instruction programs can accept a limited range of free responses in textual format.

Cognitive-Process Items

Cognitive-process items assess whether an examinee possesses a cognitive skill without, in theory, requiring the examinee to possess any knowledge. The distinction between cognitive-process and knowledge items is similar to the distinction that may once have existed between aptitude and achievement before, among other examples, vocabulary achievement became a primary component of scholastic aptitude.

Several cognitive-process tests considered for use in armed service recruit selection and placement have been listed and described by Cory

(1973, 1977, 1978). These include five tests of memory and four tests of concept formation. A "Memory for Objects" test tachistoscopically presented frames containing pictures of four to nine objects. The examinee's task was either to recall and type the items or to recognize the items from a word list. The "Memory for Words" test was similar except that the frames consisted of three to nine three- or five-letter words. The "Visual Memory for Numbers Test" presented a sequence of four to thirteen digits, one digit every second; the examinee had to recall the sequence. The "Auditory Memory for Numbers" test was similar except that the digits were presented orally by a tape recorder. The "Object-Number Test" was a paired-associates learning task.

The first concept formation test was "Computerized Twelve Questions," which was like the game Twenty Questions. It differed in that a list of possible questions was supplied from which only twelve could be chosen. "Computerized Password" presented clue words; the examinee's task was to identify the object that the clues suggested. The "Nonsense Syllogisms Test" presented syllogisms containing nonsense words. In this test, the examinee had to indicate whether the conclusion of the syllogism was correct. The "Inference Test" contained items consisting of a statement and possible conclusions; the examinee had to indicate which conclusions were correct.

Barrett, Alexander, Doverspike, Cellar, and Thomas (1982) developed a battery of information-processing tests. Their cognitive-process items (most of which measured aspects of short term memory) included three cognitive-process tests. The "Sequential Memory" test consisted of one to five letters presented sequentially. Each letter was presented for 0.8 second followed by a 0.2 second delay. The presentation of the last letter was followed by a 2 second delay, after which a probe letter was presented. The examinee responded by pressing a button on the response panel, indicating whether the probe letter was the same as or different from any one of the memory set letters. The examinee had 3 seconds to respond. The "Simultaneous Memory" test consisted of one to five letters presented simultaneously in a horizontal array for 3 seconds. These letters were erased and after a 2 second delay the probe letter was presented. The examinee responded by pressing a button on the response panel, indicating whether the probe letter was the same as or different from any one of the memory set letters. The examinee again had 3 seconds to respond. Scoring was the same as for the sequential memory test. The "Multiple Item Access" test presented two sets of letters. The first letters were presented simultaneously in a horizontal array for 3 seconds, then erased. After a delay of 2 seconds, the second five letters were presented simultaneously in a horizontal array for 3 seconds. The examinee then responded according to how many letters, one to five, were the same in the sets of letters. There were a total of 60 trials. The score was the number correct.

Church and Weiss (1980) described a spatial reasoning test that presented two matrices of fifteen numbers in a four-by-four grid. The examinee was to move the numbers of one matrix to make the matrices match by moving single numbers into the single vacant slot. Movement was accomplished by entering the number to be moved and the direction in which to move it.

The main administrative difference between the knowledge items and the cognitive-process items appears to be the requirement for a dynamic administration process in some of the cognitive-process items. The memory items require that portions of the items be sequenced with precise timing between portions. Additionally, as implemented by Cory, some of the items require free responses.

Perceptual-Motor Items

Perceptual-motor items differ from the items discussed so far in that the previous items require discrete (e.g., keyboard) responses. Perceptual-motor items may require a continuous response of some form (e.g., joystick) rather than a press of a finite number of response buttons. These items also may require a pictorial or graphic stimulus.

Cory (1973) described two tests of perceptual speed, four tests of perceptual closure, and two tests of detection of motion. The "Comparing Figures" test, one of the perceptual speed tests, presented sets of squares or circles with embedded vertical or horizontal bars. The examinee's task was to determine if all of the bars were oriented in the same way. The "Counting Numbers" test presented a string of numbers and asked the examinee to determine the frequency of a specified number.

"Recognizing Objects," one of the perceptual closure tests, presented pictures of common objects with 10 to 90 percent of the picture blotted out. The examinee's task was to identify the object. The "Gestalt Completion" test was very similar. "Concealed Words" presented words with missing letters and required the examinee to identify the word. The "Hidden Patterns" test required the examinee to recognize a test pattern embedded in a larger pattern.

The "Memory for Patterns" test, one of the movement detection tests, presented a pattern by sequentially blinking a sequence of dots. Examinees were asked to identify the patterns. The "Drift Direction" test presented a dot moving slowly by a line. The task was to determine if it was moving toward, away from, or parallel to the line.

Hunter (1978) described two computerized perceptual-motor tests that were outgrowths of earlier mechanical tests. "Two-Hand Coordination" required the examinee to move a cursor to follow a target as it rotated around the screen. The cursor was controlled by two hand controls, one for vertical movement and the other for horizontal movement. The score was computed as the average distance from the target over a five minute period. This type of item differs from those previously discussed in that it requires real-time control of the screen and an examinee interface that is mechanical and continuous rather than a discrete response.

A second test described by Hunter was "Complex Coordination." This test required the examinee to keep one cursor on target with a two-axis joystick and a second cursor centered, in one dimension, through the use of foot pedals. The required actions roughly simulated the motions required to fly an airplane. The test was scored by computing the average distance from the cursors to the targets. Administratively, this test is very similar to "Two-Hand Coordination." The primary difference is the added complexity of the stimulus and the response.

Barret et al.'s (1982) battery also included several tests of perceptual memory abilities. In the "Array Memory" test, four figures were presented simultaneously on the screen for 2 seconds and then were erased. The figures consisted of a pound sign, arrow, roman numeral five, and an "X." The examinee was then presented with one of the four figures and required to indicate on the response panel in what area of the screen that figure had previously appeared. Examinees had 3 seconds in which to respond. The score on the test was the number of correct responses. The "Vector Memory" test was identical to the above "Array Memory" test except that, after presentation of the four figures, the examinee was shown two figures and required to indicate in which area of the screen the two figures would meet if one of them moved horizontally and the other moved vertically. Examinees had 3 seconds in which to respond. The score was the number of correct responses.

The "Visual Search" test presented a probe letter for 0.8 second before it was erased. After a delay of 2 seconds, a set of one to five letters was presented simultaneously in a horizontal array for 3 seconds and then erased. The examinee responded by pressing the appropriate button on the response panel. This indicated whether the probe letter was the same as or different from any one of the memory set letters. Examinees had 3 seconds to respond. In the "Linear Scanning" test, twenty equilateral triangles were presented in a row. All but one, two, three, or four of the triangles had lines through them. The row of triangles was presented for 1.5 seconds and then erased. The examinee was then required to indicate whether one, two, three, or four of the triangles were without lines by pressing the corresponding button on the response panel. The number of triangles without lines through them varied randomly across the 20 trials. Examinees had 3 seconds in which to respond. The score was the number of correct responses. A similar "Matrix Scanning" test differed from the "Linear Scanning" test only in that the triangles were presented in a 4 x 5 matrix.

Barrett et al.'s test battery also included a reaction time test. The test presented a warning signal (an asterisk) and followed it 1 to 5 seconds later with a letter or a number. The examinee responded by pressing a button on the response panel to indicate whether a letter or a number had been presented. The score consisted of mean response time for the correct responses.

Past versions of the ASVAB have contained perceptual tests. The ASVAB-5 (Fletcher & Ree, 1976) contained an "Attention to Details Test" and a "Space Perception Test." The former was a timed clerical test that required the examinee to count "C"s embedded in a field of "O"s. The latter contained patterns representing flat sheets of paper or metal. The examinee's task was to recognize into what shape it was folded. The ASVAB-8 (Ree, Mathews, Mullins, & Massey, 1982) contained a "Coding Speed" test which required the examinee to match a word to a key and determine the corresponding number.

Simulations

Simulations are tests designed to realistically replicate specific aspects of other environments. Medium- and high-fidelity aircraft simulators are routinely used for flight training. Simulations can also be used for testing. A general-purpose testing machine should probably limit its area of capability to information-intensive simulations, however, which require presentation of information and collection of decision data. Several simulations within this portion of the domain have been reported in the literature.

Prestwood (1980) briefly described a system for administering medical simulations. Medical residents were allowed to find symptoms, administer treatments, and cure (or kill) a patient. Administratively, the items were in a multiple-choice format and the simulation proceeded by branching mechanically from a response category to another item. Implementation of simulations like these is not much more difficult than implementation of simple multiple-choice items.

A paper-and-pencil simulation of submarine component maintenance was described by Robinson and Walker (1978). Their simulation was in a multiple-choice, answer-until-correct format. Examinees used a latent-ink answer sheet that branched them to an item in a book when a correct answer was given. This simulation, unlike Prestwood's, required high-resolution representation of oscilloscope screens and equipment panels. The simulation logic could be handled easily in the same manner as Prestwood's, however.

Knerr (1978) discussed a training simulation in which the components of an electrical circuit were simulated. His purpose for using the simulation was to teach student troubleshooters to think like expert troubleshooters. Such a simulation could also have been used to evaluate the competence of student troubleshooters compared to expert troubleshooters. The advantage of such a simulation is that all possible responses do not need to be explicitly programmed. The disadvantage of such a simulation is that a simple means of defining and connecting components needs to be developed.

Non-Cognitive Items

Non-cognitive items include all items that do not assess knowledge, skill, or ability. They include such things as personality items, interest items, and value judgments. Administratively, they are very similar to knowledge items because they ask a question which, theoretically, can be answered. Any testing system capable of administering cognitive items should be capable of administering non-cognitive items.

A Survey of Testing Models

A computerized testing system probably cannot be justified if it only administers conventional paper-and-pencil tests on a computer terminal. Fortunately, psychometric technology has kept pace with computer technology. There are new testing strategies, more efficient than the old, that take advantage of the computer's high-speed computation facilities. A set of procedures and algorithms, collectively called adaptive testing methods, makes testing more efficient by selecting items to meet the assessment needs of each specific individual. In ability testing, this amounts to administering easy items to low-ability examinees and difficult items to high-ability examinees. The objective of a testing strategy is to simultaneously estimate an examinee's ability and administer items tailored to that ability.

Vale (1981) grouped most of the adaptive testing strategies developed to date into three categories: inter-item branching strategies, inter-subtest branching strategies, and model-based branching strategies. A multipurpose testing system should include the capability to administer adaptive tests of all three forms. For the inter-item and inter-subtest branching models, the difficult task in system design is to provide a convenient method of specifying the algorithms. For the model-based procedures, the greatest challenge is in selecting and designing a set of statistical programs that will allow the testing strategies to be implemented efficiently. Most current CAT research is on the model-based strategies. This category from Vale's classification has thus been expanded in the discussion below.

Inter-Item and Inter-Subtest Branching Strategies

The inter-item branching strategies are implemented by structuring an item pool so that testing begins with one item and each response to each item leads to administration of a specific item. In the typical inter-item branching strategy, a correct response leads to a more difficult item and an incorrect response leads to an easier item. Examples of this class of strategies are the pyramidal and Robbins-Monro strategies (Krathwohl & Huyser, 1956; Lord, 1971a; Weiss, 1974; Weiss & Betz, 1974).

The inter-subtest branching strategies are similar in concept to the inter-item branching strategies except that the branching is from subtest to subtest rather than from item to item. A subtest is simply a group of items. Vale (1981) further divided this class of strategies into re-entrant and nonre-entrant forms. A re-entrant form was one in which administration could branch to a subtest, out of the subtest, and back into it. An example of the nonre-entrant form is the two-stage test (Angoff & Huddleston, 1958; Weiss, 1974; Weiss & Betz, 1973) in which the score on a routing test determines the appropriate measurement test. One example of a re-entrant strategy is the flexilevel strategy (Lord, 1971b; Weiss, 1974). In the flexilevel test, items are ordered by difficulty and testing begins in the middle. If the items are split into an easy subtest and a difficult subtest, administration branches back and forth between them, branching to the difficult subtest after a correct response and to the easy subtest after an incorrect response. The stradaptive strategy (Vale & Weiss, 1978), another re-entrant form, groups items into several strata and branches to a more difficult stratum after a correct response, or to an easier stratum after an incorrect one.

A concept basic to the inter-item and inter-subtest branching strategies is the subtest module. An inter-item strategy branches among subtests that contain one item each. Inter-subtest strategies branch among subtests that contain several items. This allows the possibility of leaving the subtest to administer other items and then returning to it later. A pointer must be kept to indicate where the subtest should be re-entered.

A pyramid or a Robbins-Monro strategy is simple to set up. A structure is built such that each item response (or a correct-incorrect category set) leads to a specific item. The specification process requires only a means of specifying the branch locations. A two-stage or a multistage strategy is identical except that some means of defining subtests is required. With subtests defined, the structure specification is identical to that of the inter-item strategies.

Although it has not been done, it is reasonable to consider subtests within which the branching is non-linear. A stradaptive test with strata administered by the flexilevel strategy is an example of such a possibility. In that case, the number administered and the number correct would be required to determine where to return. Other intra-subtest branching strategies could also be considered.

Unidimensional IRT-Based Branching Strategies

Most of the currently popular strategies are model-based branching strategies. They are based on a statistical model, called item response theory or IRT (Birnbaum, 1968; Lord; 1980), which relates item responses to ability. IRT-based testing models typically administer an item based on some ability estimate and use the response to that item to improve the ability estimate. They then choose another item based on that estimate, and repeat the process for a fixed number of items or until the estimate becomes sufficiently precise.

Conceptually, model-based testing strategies are simple to administer; the item expected to most improve the estimate of ability is selected and administered, the ability is re-estimated, and the process is repeated. Practically, the difficulties arise in determining which item is expected to most improve the ability estimate and in estimating the ability. Both processes are dependent on the statistical model used and most are based on IRT.

IRT models. IRT refers to a family of psychometric theories that express the probability of choosing a particular item response as a function of an underlying trait or ability. The model most often used for CAT is the three-parameter logistic model (Birnbaum, 1968).

In the three-parameter logistic model, the item is characterized by the three parameters *a*, *b*, and *c*. Ability is characterized by a single parameter, theta. The *a* parameter is an index of the item's power to discriminate among different levels of ability. It ranges, theoretically, between negative and positive infinity but, practically, between zero and about three when ability is expressed in a standard-score metric. A negative *a* parameter means that a low-ability examinee has a better chance of answering the item correctly than does a high-ability examinee. An *a* parameter of zero means that the item has no capacity to discriminate between different levels of ability (and would therefore be useless as an item in a power test). Items with high positive *a* parameters provide sharper discrimination among levels of ability and are generally more desirable than items with low *a* parameters.

The *b* parameter indicates the difficulty level of an item. It is scaled in the same metric as ability and indicates what value of theta an examinee would need to have a 50-50 chance of knowing the correct answer to the item. This is not, however, the level of theta at which the examinee has a 50-50 chance of selecting a correct answer if it is possible to answer the item correctly by guessing.

The *c* parameter indicates the probability with which a very low-ability examinee would answer the item correctly. It is often called the guessing parameter because it is roughly the same as the probability of answering the item correctly if the examinee does not know the answer and guesses at random. Intuitively, the *c* parameter of an item should be the reciprocal of the number of alternatives in the item. Empirically, it is typically somewhat lower than this.

All four parameters enter into the three-parameter logistic test model to determine the probability of choosing a correct response. The formal mathematical relationship is given by Equation 1:

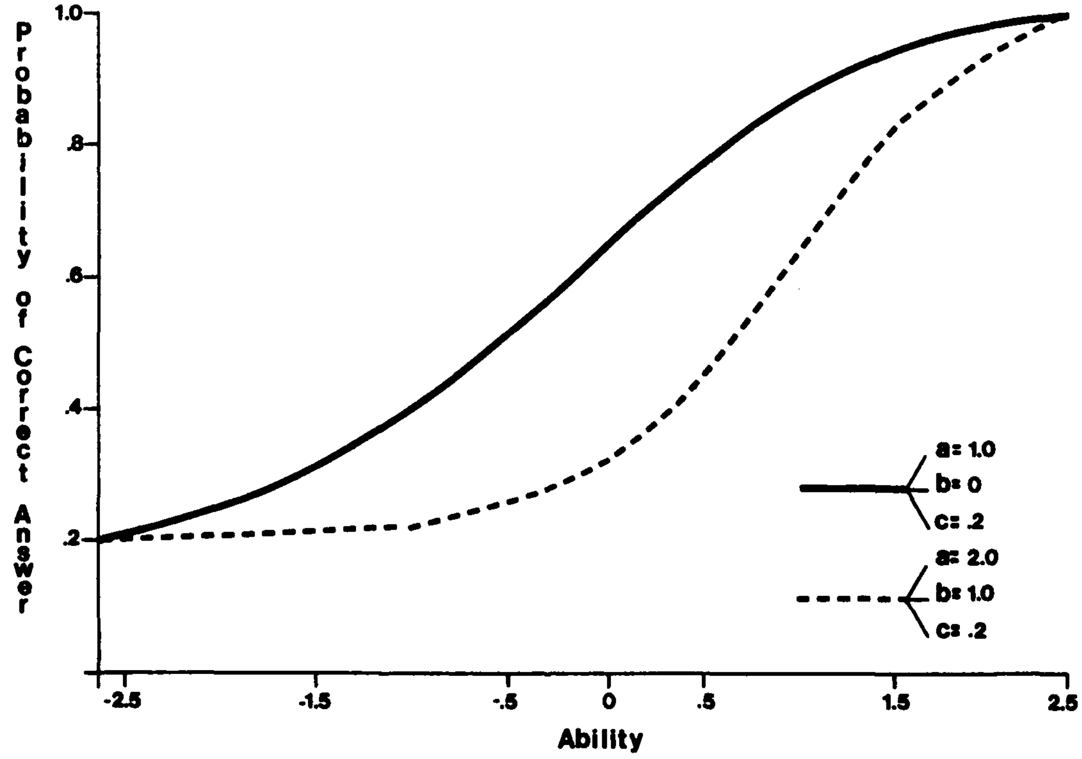
$$P(u=1|\Theta) = c + (1-c)\ \Psi[1.7a(\Theta-b)] \qquad [1]$$

where:

$$\Psi(x) = 1/[1+\exp(-x)]$$

In Equation 1, _u_ = 1 if the response to the item is correct and _u_ = 0 if the response is incorrect. The relationship expressed in Equation 1 is shown graphically in Figure 1. The item characteristic curve drawn with a solid line is for an item with _a_ = 1.0, _b_ = 0.0, and _c_ = .2. The slope at any point is related to _a_. The lower asymptote corresponds to a probability or _c_ of 0.2. The item characteristic curve shown with a dashed line is for an item with _a_ = 2.0, _b_ = 1.0, and _c_ = 0.2. The midpoint of the curve has shifted to Θ = 1.0. The slope of the curve is steeper near Θ = _b_. The lower asymptote of the curve remains, however, at 0.2.

Figure 1. Item Characteristic Curves



Several other models can be obtained by making minor modifications to the basic three-parameter model. If the guessing parameter (_c_) is set to zero, it becomes the two-parameter logistic model. If the discrimination parameter is also set to a constant, it becomes equivalent to the one-parameter or Rasch model. If the cumulative-logistic function is changed to a cumulative-normal function, it becomes a one-, two-, or three-parameter normal-ogive model. Computationally, the logistic and normal-ogive models are nearly equivalent. Scoring is simpler in models with fewer parameters.

Samejima (1969) suggested an extension of the dichotomous model to what she called the graded-response case. The graded model is useful for categorical-response items in which the categories are ordered in difficulty or trait level. This model might be used for answer-until-correct ability items, like-indifferent-dislike interest items, or multipoint rating-scale items. In this model, the bounds between adjacent categories are scaled along the trait continuum using one of the dichotomous response models. The characteristic curves of the upper boundaries then refer to the probabilities of correctly endorsing a category at least that far toward the upper end of the continuum. In the case of an answer-until-correct item, where the categories are the number of choices needed to choose the correct one, the upper bound for the category of "three tries" would indicate the probability of answering the item correctly in three or fewer tries. The probability of answering the item correctly in exactly three tries (i.e., in the "three tries" category) would be the probability of answering correctly in three or fewer times minus the probability of answering correctly in two or fewer tries.

Samejima proposed both a homogeneous and a heterogeneous model for graded-response data. In the homogeneous case, all response categories were required to discriminate equally well at their respective locations on the ability continuum. This assumption was relaxed in the heterogeneous model. The advantage of the heterogeneous model is its ability to accommodate a wider range of items. The disadvantage of the heterogeneous model is that it requires estimation of more parameters for the items.

A model mathematically very similar to Samejima's heterogeneous model was proposed by Bock (1972) for the nominal-response case. In this case, the response categories were not assumed to be ordered, although ordered categories would also fit the model. Bock's model was developed only for use with the logistic-ogive function. Samejima's could, theoretically, be used with either the logistic or the normal-ogive functions.

Samejima (1979), dissatisfied with certain characteristics of available IRT models for use with multiple-choice items, developed a new family of models. Specifically, she extended the graded models to include a random response category for examinees who had no idea what the correct answer to the item was. These models are a theoretical improvement over the three-parameter logistic model because they extract additional information from the item's distractors and always result in a unique ability estimate (the three-parameter logistic models do not). They require several more parameters for each item and, therefore, require somewhat more extensive computational effort.

One goal of fair testing is to measure ability with equal precision at all levels. Toward this goal, Samejima (1980) developed a constant information model that provided a constant level of information (and thus equal precision) over a range of ability. Theoretically, such items could be combined more easily into an equiprecise test than items calibrated according to other IRT models. Practically, it has yet to be demonstrated that the model adequately characterizes any useful item type.

Ability estimation. In an IRT-based adaptive testing strategy, scoring (or ability estimation) typically takes one of three forms: least-squares Bayesian, modal Bayesian, or maximum-likelihood. All three methods are based on the test likelihood function. In IRT, the item responses are assumed to depend only on the underlying ability. When that is fixed, item responses are assumed to be independent. The item characteristic function expresses the probability of choosing a response as a function of ability. When several items are administered, the item response characteristic functions can be multiplied together resulting in a probability of the overall response pattern occurring at any given point. When these points are considered as a function, it is called a likelihood function.

The maximum-likelihood estimate of ability (Birnbaum, 1968) is the point on the ability continuum corresponding to the maximum of the likelihood function. It is computed by finding the root of the first derivative of the logarithm of the likelihood function. The log-likelihood function is used, rather than the likelihood function, for computational simplicity.

Modal Bayesian estimation (Samejima, 1969) assumes prior knowledge of the distribution of ability. A standard normal prior, for example, might be appropriate if, in a relevant population, ability is distributed normally with a mean of zero and a standard deviation of one. Modal Bayesian scoring is accomplished by multiplying the prior probability-density function and the likelihood function together and finding the ability level corresponding to the maximum. Practically, the logarithm of this joint function is used.

Least-squares Bayesian estimation (Owen, 1969, 1975; Vale, 1980) is accomplished using the same joint function used for modal-Bayesian scoring. The mean of the function rather than its mode is used in the estimation process, however.

Item selection. Using an IRT-based strategy, item selection minimizes the error of measurement following administration of each item. In the Bayesian strategies, this may be done by minimizing the expected variance of the posterior distribution (Owen, 1969, 1975). In the maximum-likelihood strategy, it is usually done by maximizing the local item information (Birnbaum, 1968). Item information, because of its greater ease of computation, is sometimes used in the Bayesian strategies too. These two goals are accomplished, conceptually at least, by evaluating the information or posterior variance expected to result from each item and then selecting the best item. Practically, more efficient procedures (e.g., Vale & Weiss, 1977) are used.

Multidimensional IRT-Based Strategies

Unidimensional IRT requires that all test items measure a single trait. When batteries of tests are administered using unidimensional IRT, they must consist of subtests, each composed of unidimensional items. Two general approaches to the efficient administration of multidimensional batteries have been taken: the fully multidimensional and the semi-multidimensional approaches.

Fully multidimensional testing allows the individual items to assess different latent dimensions (Samejima, 1974; Sympson, 1978) or to consider multiple cognitive components (Whitely, 1981). These models allow the most latitude in the types of items that can be accommodated. They would be extremely difficult to implement because of the computational requirements. Design of a testing system probably does not need to consider them, however, since computational difficulties at the item calibration stage has precluded any practical implementation of these models.

The semi-multidimensional strategies (Brown & Weiss, 1977; Vale, 1980) have been practically implemented, however. The semi-multidimensional strategies, like the unidimensional strategies, require test items to be grouped into unidimensional subtests. Information from item responses in each subtest is used to tailor the items in the other subtests. In the Brown-Weiss procedure, one subtest is completed before proceeding to another. The various subtests in Vale's procedure may be administered simultaneously, branching from one to another and back again, as required. The Brown-Weiss procedure is computationally no more difficult to implement than a unidimensional strategy. Vale's procedure is somewhat more difficult because it considers items from all content areas as each item is selected.

Order-Theory Strategies

Adaptive testing based on order theory is a relatively new strategy. As described by Cliff (1975), order theory seeks to determine a complete ordering of people and stimuli from an incomplete set of data. Basic to the concept is the dominance matrix. An examinee is said to dominate an item if s/he answers it correctly. An item is said to dominate an examinee if the examinee answers it incorrectly. A binary matrix of dimensions (persons + items) X (persons + items) can be constructed from these binary dominance relationships. By powering this dominance matrix, a complete matrix of dominance relationships can be recovered from a partial matrix of relationships, if a sufficient set of relationships exists. Given sufficient random dominance relationships, the complete matrix can be recovered. "Sufficient" will typically be more than a minimum number of relationships, however. The goal in interactive testing based on order theory is to complete the dominance matrix with a minimally sufficient set of dominance relationships.

The mathematical operations involved in accomplishing this task are formidable. Fortunately, computer programs have been developed to perform them. TAILOR (Cudeck, Cliff, & Kehoe, 1977) is an interactive program for simultaneously ordering a set of examinees and items. TAILOR-APL (McCormick & Cliff, 1977) is an interactive program for determining the order of a single examinee after a group of examinees and a set of items have initially been simultaneously ordered.

Mastery Testing Procedures

Mastery or achievement testing typically has one of three objectives: (1) determination of an individual's level of achievement or trait status, (2) classification of an individual into a category (i.e., master or non-master), or (3) determination of the change occurring in an individual's trait status over a period of time. Procedures for the first objective are identical to those used for ability testing and will not be discussed here.

Both conventional and IRT-based procedures have been used for mastery-classification decisions. The classical procedures are typically based on Wald's (1947) sequential probability ratio test, which was developed to detect breakdowns in production equipment. Wald's procedure is a likelihood ratio test applied to a sequence of Bernoulli trials. It provides a basis for making classification decisions with a specified error tolerance. As applied to test items, it is unrealistic because it assumes they are all identical. Wald's procedure has, nevertheless, been expanded into testing strategies.

Ferguson (1970) used the procedure to determine mastery position in a hierarchy of objectives. A hierarchy of learning objectives was first explicated. Then tests were developed to assess mastery at each mode in the hierarchy. Within a modal test, Wald's procedure was used to make a mastery classification in as few items as possible. Computationally, Ferguson's procedure presents no difficulties to a testing system and can probably be handled within the context of other strategies discussed thus far.

Kalish (1980) extended Ferguson's strategy by using archival data to predict responses to unadministered items from a limited set of administered items. Specifically, he attempted to find, for an examinee, a matching sequence of items in the archival database and then make predictions about items yet to be administered. Kalish's strategy could be quite difficult to administer efficiently because it requires a database organized by unique response vectors and this could require a large amount of storage.

Kingsbury and Weiss (1979) developed a mastery-testing extension of Owen's (1969, 1975) Bayesian testing strategy. The testing procedure was identical to that used for measurement (as opposed to classification). However, the procedure terminated when the decision regarding whether the examinee was above or below a cutoff could be made with less than a specified amount of error. Administratively, this procedure is only slightly more difficult to implement than the original Owen strategy.

A Survey of Potential System Users

Interviews with Potential System Users

Twenty-seven individuals from seven different organizations were interviewed to gain insight into potential testing needs and problems not apparent from a review of the literature. The organizations included Educational Testing Service; Psych Systems, Inc.; the Navy Personnel Research and Development Center, San Diego; the Air Force Human Resources Laboratory, San Antonio; the American College Testing Program; the Naval Training Center, Great Lakes; and Continental Illinois National Bank. This sample represented a relatively wide range of testing applications ranging from individual testing to testing of large groups, and from selection testing to course-achievement testing.

Each person interviewed was given a brief description of the project and its objectives as well as a preliminary design for a microcomputer testing system. The design presented listed a set of system components including a processor, a disc, a terminal, and testing software. The interviewee was then asked to suggest what special characteristics a testing system might require for his/her application. S/he was further asked if s/he had any experience with similar systems and, if so, what the strengths and weaknesses of these systems were. The highlights of the interview responses are presented in five areas below.

System hardware. Comments about hardware fell into three areas: analog input, security, and external interfaces. The most common form of analog input desired was the joystick. There was some concern regarding cost, however, because joysticks of laboratory quality can cost $2,000.00. Light pens were not considered an acceptable form of input, primarily because examinees tend to chew on them and thereby destroy them. Touch screens were considered a good alternative to light pens when they provided sufficient response resolution.

Theft was a major concern in the area of machine security. The suggestion was made that the testing systems be equipped with an alarm and, further, that the storage media (e.g., discs) have some means to prevent people from reading them on other systems. In addition, there was some concern that examinees would "experiment" and press buttons they should not touch or even pull floppy discs out of drives if they were part of the testing terminal. Outright vandalism was not considered to be a serious problem.

Several potential interface requirements were suggested. The most common was the videodisc interface. Others included special-purpose devices such as a desk-top flight simulator.

Test authoring. Several of the individuals interviewed had experience with either computerized testing systems or with courseware development systems. Most of the courseware authoring had been done using an author language. Most test development had been done with special-purpose programs written in a standard programming language. Some interviewees noted that it was difficult to get people with little or no programming experience to develop tests or courseware using an authoring language.

Potential applications. Several potential applications for a microcomputer testing system were suggested. These included clinical assessment of personality and psychopathology, individualized selection testing, computer-assisted-instruction and achievement testing, and computerized counseling. A stand-alone microcomputer system did not appear to be applicable to large-scale group testing (e.g., for college admissions).

Item types. Most of the item types cited in the interviews were discussed previously in the literature review. The only new type mentioned was one with sequentially dependent keying. In such an item, the correct response to an item is dependent on the examinee's responses to previous items. For example, in a medical simulation "hold for further observation" may be an acceptable response unless the examinee has, in a previous item, administered a drug that stopped the patient's heart. The sequentially dependent item provides a distinct challenge in developing a testing system.

Testing strategies. No new testing strategies were suggested in the interviews. Interviewees who were engaged in achievement testing generally were content with a system that allowed inter-item branching. Although some interest was expressed in newer models, most interviewees interested in adaptive testing had no immediate plans for testing strategies based on anything other than one of the logistic models.

A Questionnaire Survey of Potential System Users

A questionnaire was developed based on ideas obtained from the literature review and through interviews with CAT researchers. The questionnaire, reproduced in Figure 2, contained 28 questions grouped into four sections. The first section, which contained nine questions, focused on test characteristics and was to be answered by individuals who are responsible for test development. The second section contained nine questions and focused on test administration; answers to these questions were solicited from individuals who were responsible, directly or indirectly, for test proctoring. The third section, System Procurement, asked how much a system should cost and what features it should have. It was intended for individuals who were responsible for purchasing or ordering systems such as this. The fourth section, Test Development, was intended to assess the familiarity with computers of those individuals who would be involved in the actual implementation of tests on the system.

A list of potential system users was assembled from various mailing lists and lists of participants at conferences that emphasized the use of computers in testing. The final list contained 108 names. Questionnaires were mailed to all of these individuals. Each package contained a questionnaire, a stamped addressed return envelope, and a personalized cover letter explaining the project and the purpose of the questionnaire. The letter asked that the questionnaire be returned within one week.

By the end of the third week after the original mailing, 40 completed questionnaires had been returned. In addition, 10 individuals sent notes or letters indicating that they did not feel that they should respond to the questionnaire because they were not directly involved with testing or because they did not want to influence the results of the survey. Five more questionnaires were returned by the post office because the addressees had moved. By the end of the third week, 55 questionnaires had been accounted for.

A follow-up letter, along with an additional questionnaire, was sent to the remaining individuals. This resulted in ten more completed questionnaires. Analyses were performed on the 50 completed questionnaires.

Individuals usually responded to some but not all sections of the questionnaire. This was expected because not all sections applied to all individuals. Missing data were handled systematically for each item, but the method differed from item to item. In general, there were two methods of deciding if data were missing. Unless otherwise noted in the text, if an individual responded to any item within a section, s/he was assumed to have responded to all items in the section. Thus, the percentage of endorsements was computed using the number of individuals responding to the section as the denominator of the fraction. The exception to this was in the computation of means of numbers (such as the length of a test) where the data were considered missing if the item was left blank.

Test Characteristics. The first three items in the Test Characteristics section dealt with test length. Forty-two individuals responded to the first question about the length of their longest test. Lengths ranged from 16 to 1,000 items. The mean of the forty-two responses was 138. The mean is not particularly informative for system design, however, because a system must be able to accommodate unusual cases. Percentile points were thus calculated for the seventy-fifth, ninetieth, and ninety-fifth percentile ranks. For this calculation, percentile points were defined as the value at or below which the specified percentage fell. Thus, a system designed for the seventy-fifth percentile point would accommodate 75% of the respondents. The seventy-fifth, ninetieth, and ninety-fifth percentile points for the first question were 150, 250, and 329 items, respectively. As with the means, these percentile ranks were calculated using only the individuals responding to the item.

Thirty-one individuals responded to the second question about how many items their adaptive pools would contain. The mean response was 1441 items and the three percentiles were [illegible]000, 2500, and 10000 items. Thirty-seven individuals indicated the average length, in English words, of the items on their longest test. The mean length was 109 words and the percentiles were 150, 326, and 400. Thirty individuals responded to both the pool-size and the item-length questions. The pool size, in words, was taken to be the product of these two answers. The average of this product was 160,700 words. The three percentiles were 90,000; 180,000; and 1,200,000 words.

The remainder of the first section was concerned with the item types and the scoring and item-selection algorithms that were required by the respondents. Forty-four individuals responded to this section. Asked whether any items required a timed stimulus, 59% responded that none did. A simple timed display was required by 36% and a moving display was required by 5%. Only 14% said that their items did not include any pictures. Eighty percent said their items contained line drawings, 41% said they included shaded drawings, and 9% said they contained moving pictures.

In response to the item asking what item-selection strategies they used or might consider using in the future, 91% indicated that they would use conventional tests, 36% said that they would use mechanically branched adaptive tests, 68% said that they would use statistically branched adaptive tests, and 18% said that they would use implied-order-based branching. For conventional scoring methods, 80% said they would use number-correct scoring and 46% said they would use formula scoring. For unconventional methods, 50% said that they would use maximum-likelihood scoring, 52% said that they would use Bayesian scoring, and 9% said that they would use implied-order-based scoring.

The final question asked what item-response-theory scoring methods they would use. Thirty percent said that they would not use any. Thirty-nine percent said that they would use Rasch scoring.

FIGURE 2. COMPUTERIZED TESTING SYSTEM QUESTIONNAIRE

COMPUTERIZED TESTING SYSTEM QUESTIONNAIRE

TEST CHARACTERISTICS — ANSWER THE QUESTIONS IN THIS SECTION IF YOU ARE RESPONSIBLE FOR SELECTING OR DEVELOPING TESTS.

If you use conventional fixed-length tests, how many items are included in your longest test? ____ Items

If you use or plan to use adaptive or tailored tests, how many items will be in your largest item pool? ____ Items

How many English words, on the average, do each of the items in your longest test or item pool contain? ____ Words

Do any of your tests require a timed presentation of the item stimulus?
- No ____
- Yes (what kinds):
  - An unchanging, timed display ____
  - A moving display (eg, pursuit rotor) ____

Do any of your items include pictures?
- No ____
- Yes (what kinds):
  - Line drawings ____
  - Shaded pictures ____
  - Moving pictures ____

What item-selection methods or testing strategies are you currently using or might you consider using in the future?
- Conventional tests (fixed item sequence) ____
- Adaptive tests (what kinds):
  - Mechanical branching (eg, pyramidal) ____
  - Statistical branching (eg, maximum-information) ____
  - Implied-order-based branching ____

What scoring methods are you currently using or might you consider using in the future?
- Conventional number-correct scoring ____
- Conventional formula scoring ____
- Maximum-likelihood scoring ____
- Bayesian scoring ____
- Order-theory scoring ____

What Item-Response-Theory models underlie the testing and scoring procedures that you are currently using or might use in the future?
- None ____
- Rasch ____
- 2-parameter dichotomous response ____
- 3-parameter dichotomous response ____
- Nominal polychotomous response ____
- Graded polychotomous response ____
- Continuous response ____

TEST ADMINISTRATION — ANSWER THE QUESTIONS IN THIS SECTION IF YOU WILL SUPERVISE PROCTORS OR BE RESPONSIBLE FOR PROCTORING THE ADMINISTRATION OF EXAMINATIONS ON THE SYSTEM.

What are the general ability levels of the examinees you will test?
- Pre-school ____
- Grade school ____
- High school ____
- College ____

What is the longest delay you can tolerate between an examinee's response and the presentation of the next item?
- Less than 1 second ____
- 1-2 seconds ____
- 2-5 seconds ____
- More than 5 seconds ____

Will a proctor be readily available to assist examinees having trouble with the test or the testing system?
- Usually ____
- Occasionally ____
- Rarely or never ____

How should the system signal the proctor that an examinee is having trouble?
- A buzzer on the examinee's terminal ____
- A light on the examinee's terminal ____
- A message on a proctor's status screen ____

What conditions should the system be able to detect while monitoring an examinee?
- Unreasonable response delays ____
- Random responding ____
- Aberrant response strings ____
- Other (Identify) ____________

How much wear and tear will the examinees inflict on the system?
- They may inflict deliberate damage (eg, kick it). ____
- They may inflict accidental damage (eg, spill things on it). ____
- They may punch buttons or pull levers harder than necessary. ____
- They will probably not be unnecessarily rough with it. ____

In a typical month, how far is the equipment likely to be transported?
- It will rarely or never be moved. ____
- From one room to another within a building ____
- From one building to another within a city ____
- From one city to another ____

Should the system prevent unauthorized access?
- Yes ____
- No ____

Should test information be encoded so that it can not be read on another computer?
- Yes ____
- No ____

SYSTEM PROCUREMENT — ANSWER THE QUESTIONS IN THIS SECTION IF YOU COULD INITIATE A REQUEST TO PURCHASE ONE OR MORE SYSTEMS IF YOU NEEDED THEM.

How many basic testing systems would you consider buying at the following prices?

Basic test administration system (everything you need -- CPU, CRT, disk drive, and *software* -- to administer tests that have textual questions and no graphics)

Under $2,000 ____ systems
$2,000 to $3,000 ____ systems
$3,000 to $5,000 ____ systems
$5,000 to $10,000 ____ systems

Basic test development system (everything you need to develop and administer tests that have textual questions and no graphics -- includes above equipment plus extra *disk* drive and *software*)

Under $2,000 ____ systems
$2,000 to $3,000 ____ systems
$3,000 to $5,000 ____ systems
$5,000 to $10,000 ____ systems

Which of the following options, at the prices *listed* in parentheses, would you like to include in some of the basic systems?

Display Options
High-resolution digital
Black & white line drawings ($1,000) ____
Color -- text only ($1,000) ____
Color line drawings ($2,000-$4,000) ____
Shaded color pictures ($6,000) ____
Video tape or disc pictures ($1,000) ____

Input Options
Simplified keyboard ($200) ____
Joystick, trackball, or similar ($200-$400) ____
Touch screen ($500-$1,200) ____
Voice recognition ($500-$5,000) ____

Output Options
Musical tone generator ($200) ____
Voice synthesizer ($100-$1,000) ____

General Software Options
Compilers ($200-$400) ____
Accounting systems ($200-$1,500) ____
Word processing ($200-$800) ____

We may have some prototype systems that we would like to field test. Would you be interested in trying one of them if it were furnished free of charge?
Yes ____
No ____

TEST DEVELOPMENT — ANSWER THE QUESTIONS IN THIS SECTION IF YOU WILL SUPERVISE OR BE RESPONSIBLE FOR IMPLEMENTING TESTS AND TESTING STRATEGIES ON THE COMPUTER SYSTEM.

Have you run computer programs?
No, I have never run computer programs. ____
Yes (what kinds of systems):
Large Computers (eg, IBM 360, Cray 1) ____
Minicomputers (eg, PDP 11, HP 1000) ____
Microcomputers (eg, Pet, Apple) ____

Have you written computer programs?
No, I have never written computer programs. ____
Yes (what kind of programs):
Statistical using packaged programs (eg, SPSS) ____
Applications using BASIC, Pascal, or FORTRAN ____
Applications using an assembly language ____
Courseware using an author language (eg, PILOT) ____

Do you prefer to develop tests using an author language or a menu system? (See the note below if you are not familiar with the difference.)
Prefer author language ____
Prefer menu system ____
No preference ____

NOTE : There are two general methods of specifying how a test is to be structured and administered. One is through an author language; the other is by using a menu system. An author language is a special-purpose programming language, which is easier to use than a general-purpose language like FORTRAN. An author language allows a test developer to write a testing program which, when executed, administers a test. A menu system asks questions and the test developer supplies answers. A menu system is typically easier to use than an author language but is somewhat less flexible.

WHEN YOU HAVE COMPLETED THIS QUESTIONNAIRE:

Thank you for taking the time to complete this questionnaire. Providing your name on the line below is optional, but it will help us to determine who has returned the questionnaire and prevent us from sending you unnecessary reminders to do so. Whether or not you choose to give your name, we thank you again for helping us with our survey.

Name ______________________

Assessment Systems Corporation
2395 University Avenue, Suite 306
St. Paul, Minnesota 55114

Twenty-seven percent said that they would use the two-parameter logistic model and 71% said that they would use the three-parameter logistic model. For the more sophisticated response models, 23% said that they would use the nominal logistic model, 18% said that they would use the graded logistic model, and 21% said that they would use the continuous logistic model.

Test administration. The second section of the questionnaire addressed the practical considerations of everyday administration. Forty-three individuals responded to this section.

The educational levels of examinees for which the system would be used were primarily high school and college. One respondent said that the system would be used at the pre-school level, 26% said that it would be used at the grade-school level, 77% said it would be used at the high-school level, and 65% said it would be used at the college level.

Forty individuals responded to the question about the desired length of the delay between an examinee's response and the presentation of the next item. Of these forty, 8% said a delay of over five seconds would be tolerable, 33% said a delay of two to five seconds would be tolerable, 43% said a delay of one to two seconds would be tolerable, and 16% said a delay of less than a second would be tolerable.

Eighty-eight percent said a proctor would usually be available to assist the examinee, 12% said a proctor would occasionally be available, and only one respondent said a proctor would rarely or never be available.

Seventy-seven percent said that the system should signal that an examinee was having difficulty by sending a message to a proctor terminal. Twenty-eight percent said that a light on the examinee's terminal would be acceptable. Sixteen percent said that a buzzer on the examinee's terminal would be acceptable.

When asked what problems the system should be able to detect automatically, 84% said it should detect unreasonably long delays in responding, 56% said it should detect random responding, and 62% said it should detect aberrant (i.e., unusual) response strings.

When asked how much abuse the system might receive, 26% of the respondents felt that the examinees would not be unneccessarily rough with it, 67% thought that they might push buttons or pull levers harder than necessary, 54% thought that they might inflict accidental damage such as spilling things on it, and 23% thought that they might inflict deliberate damage on it.

Fifty-six percent of the respondents said that the system would rarely have to be moved once it was installed. Twenty-eight percent said that it would be moved from room to room, 19% said it would be moved from building to building, and 19% said it would be moved from city to city.

Regarding system security, 91% said the system should have some feature to prevent unauthorized use and 63% felt that the system should encode the test information to prevent it from being read on another computer.

System procurement. The third section attempted to determine what features would be desirable, when weighed against their cost. This section was designed to be completed by individuals who were in a position to buy or authorize the purchase of a system. System features were listed along with their approximate price and respondents were asked which features they would like to buy.

The first two questions dealt with basic systems. The first listed the basic test administration system, capable of administering but not developing tests. Four price ranges were listed (under $2,000; $2,000 to $3,000; $3,000 to $5,000; and $5,000 to $10,000) and respondents were asked to indicate how many of each they would consider buying at each of the price ranges. The second asked the same question about a system capable of developing tests. Responses to these items were rather difficult to interpret because of some aberrant response tendencies. First, some individuals responded with an X instead of a number; the X was replaced with a 1 in these cases. Also, some individuals indicated that they would buy more systems at the higher price than at the lower prices; in these cases it was assumed that, if they were willing to buy a certain quantity at a high price, they would be willing to buy at least that many at a lower price and their responses were adjusted accordingly.

In response to the first question regarding the basic system, the 38 respondents indicated that they would buy 4,240 systems at $2,000 or less. However, they would buy only 4,208 systems if the price were $3,000, and they would buy 3,532 for $5,000 each. Respondents indicated that they would buy 3,080 systems if the cost were $10,000. These figures for the test development system were 3,484; 3,467; 3,056; and 3,041. These values must be interpreted with caution in both cases, however, because a single respondent indicated a desire to purchase 3,000 systems; this number overwhelms the responses of the remaining respondents. The totals of 1,240; 1,208; 532; and 80 for the basic administration system and 484, 467, 56, and 41 for the test development system may be more appropriate. The second set of figures shows a sharp break as the system costs rise above $3,000 and may suggest a target price for the system.

The remaining questions dealt with specific features. Seventy-four percent wanted black-and-white line drawings, 29% wanted color line drawings, 24% wanted shaded color pictures, and 53% wanted video tape or disc pictures. Only 8% needed color text displays. For input methods, 71% wanted a simplified keyboard, 32% chose an analog device such as a joystick, 50% wanted a touch screen, and 8% wanted voice recognition. Eighteen percent could use for a musical tone generator and 29% wanted a voice synthesizer. Sixty-three percent wanted the system to support high-level language compilers, 40% needed word-processing capabilities, and 13% thought an accounting system might be useful.

Eighty-four percent of the respondents indicated that they would be willing to try the testing system if it were made available to them free of charge.

Test development. The fourth section attempted to assess the familiarity of potential test developers with computers. Three questions were asked and 49 individuals responded.

The first question asked what types of computer systems the respondent had used. Only 4% (two respondents) had never used a computer. Eighty-four percent had used a mainframe computer, 67% had used a minicomputer, and 67% had used a microcomputer.

The second question asked if they had written computer programs. Six percent had not written programs; 80% had written package programs (e.g., SPSS); 92% had written BASIC, Pascal, or FORTRAN programs; 33% had written assembly language programs; and 16% had written courseware.

The final question described menu and author language methods of test specification and asked which they would prefer. Thirty-nine percent preferred the author language method, 29% preferred the menu system, and 33% had no preference.

Recommended System Characteristics

From the information drawn from the questionnaire and the literature review, the general characteristics desired for a microcomputer-based testing system can be described. These characteristics have been grouped into four areas. The first considers the characteristics neccessary for the system to store and administer the types and quantities of test items required by the applications considered. The second includes the characteristics required to ensure that computerization of testing is cost effective and convenient when compared to conventional paper-and-pencil testing. The third considers what features can be included in the system without making the cost prohibitive. The fourth considers means by which a user can create tests on the system.

Design decisions made from the literature were, in general, qualitative and intuitive. Decisions made from the questionnaire data were guided by the percentage of individuals desiring a particular system feature. The cutoffs used for the various features in making system design decisions are usually somewhat arbitrary. In this analysis, an attempt was made to set the cutoffs liberally enough to allow a majority of people to use the system while, at the same time, keeping them restrictive enough to allow a cost-effective system to be designed. For the initial system specification, the cutoffs were set at the seventy-fifth percentile for ranked variables such as the amount of storage required. Features were considered valuable if desired by at least 10% of the respondents.

Accommodation of Tests

Storage requirements. To determine computer storage requirements, each word is considered to be five characters long followed by a space. The storage requirements neccessary to accommodate the item banks of the respondents can be computed by multiplying the number of words by six. The computed average length was 964,200 characters. The seventy-fifth, ninetieth, and ninety-fifth percentiles were 540,000; 1,080,000; and 7,200,000 characters, respectively.

Using the seventy-fifth percentile cutoff, the system must be able to store a bank of 540,000 characters. This must be on-line storage because it represents the item bank used by an on-line adaptive test. The conventional tests described by the respondents are all smaller than this, so there are no further requirements for conventional tests. The system will need sufficient additional storage for the system programs and scratch files and this must be included in the design. A minimum of 100,000 characters should be allowed for this.

The resulting figure of 640,000 characters does not include storage for graphic items. Depending on the number of pictures and types of compression possible, graphic storage could greatly increase the storage requirements.

Display requirements. Table 1 presents an analysis of the required and desirable stimulus presentation characteristics for each of the item types, as determined from an intuitive analysis of the item types reviewed. X's in the table indicate the required characteristic and O's indicate desirable characteristics.

Six characteristics are considered and listed across the top of the table. Character presentation means that the system will have to display standard alphanumeric characters. Graphic display indicates the system will have to present drawings. A timed stimulus is one whose presentation time can be controlled. A real-time stimulus is one that may have several timed presentations. A dynamic stimulus is one that has motion within a frame rather than a series of timed frames. Timed and real-time are considered to be on a continuum; if an item requires real-time presentation, it also requires timed presentation. An auditory stimulus consists of voice, music, or any stimulus that must be heard.

The total number of X's and O's is presented at the bottom of the table and gives some indication of the need to include each of the stimulus characteristics in a multipurpose testing system. Of the 30 item types considered, 24 required character presentation of items. Graphic presentation was required by 10 and desirable for 11 more. Timed display was required by 10 item types and desirable for six more. Ten item

Table 1. Stimulus and Response Requirements for Various Item Types

	Stimulus Requirements						Response Requirements										
	Character	Graphic	Timed	Real time	Dynamic	Auditory	Keyboard	Light Pen	Touch Screen	Joystick	Trackball	Mouse	Wheel	Timed	Real time	Dynamic	Voice
Knowledge																	
Dichotomous	X	O	O			O	X		O					O			O
Answer until correct	X	O	O			O	X		O					O			O
Confidence weighted	X	O	O			O	X		O					O			O
Probabalistic	X	O	O			O	X		O					O			O
Free Response	X	O	O			O	X		O					O			O
Cognitive Process																	
Memory	X	X	X	X		X	X		O					X			
Concept	X	X					X		O					X			
Spatial Reasoning	X						X		O					X			
Sequential Memory	X	O		X		O	X		O					X			
Simultaneous Memory	X	O		X			X		O					X			
Multiple Item Access	X	O		X			X		O								
Perceptual-motor																	
Perceptual Speed	X	X	X				X		O					O			
Perceptual Closure	X	X	O				X		O					O			
Movement Detection		X	X	X	X		X		O					O			
2-Hand Coordination			X	X	X					X				X	X	X	
Complex Coordination			X	X	X					X				X	X	X	
Attention to Detail	X						X							X			
Spatial		X					X		O					X			
Coding Speed	X						X							X			
Array Memory	X	O	X				X		O					X			
Vector Memory	X	O	X				X		O					X			
Visual Search	X			X			X		O					X			
Linear Scanning		X	X				X		O					X			
Matrix Scanning		X	X				X		O					X			
Choice Reaction Time	X	O		X			X		O					X			
Simulations																	
Prestwood	X						X							O			O
Robinson & Walker	X	X					X		O					O			O
Kneer	X	X					X		O					O			O
Pine	X		X	X	O		X		O					O	O		O
Non-Cognitive																	
Any	X					O	X		O								O
TOTAL																	
Required	24	10	10	10	3	1	28	0	0	2	0	0	0	16	2	2	0
Desireable	0	11	6	0	1	7	0	0	25	0	0	0	0	12	1	0	10

Note: X=required, O=desirable

types required real-time display. Dynamic display was required by three types and desirable for one more. Auditory presentation was necessary for one type and desirable for seven others.

Although this is only a rough initial estimate of system stimulus requirements, it appears that a multipurpose testing system will require both character and graphic display capabilities. Timed and probably real-time display capabilities will also be required. A dynamic display would be useful for at least a few item types. Auditory presentation capabilities might be useful but would rarely be required.

The questionnaire analyses confirm most of these conclusions. Using the cutoffs discussed above, the questionnaire data suggest that graphic displays, both line and shaded drawings, are required. A timed display is essential but dynamic presentation is not. A color display should be available, as should a videodisc interface. Finally, a musical tone generator and a voice synthesizer should be available as an option.

Response requirements. Table 1 also presents the required and desirable response collection requirements for the 30 item types reviewed. Eleven response characteristics are listed across the top of the table. A keyboard refers to either a standard CRT keyboard or a limited response keyboard including only the buttons required for response to test items. A light pen is a device for pointing to a CRT screen. To respond to touch without requiring a light pen, a touch-screen CRT is needed. A joystick is a one- or two-axis analog control stick, similar to the control stick in an airplane. More sophisticated is the trackball, a free-axis rotating ball rotated in the direction in which one desires the cursor to move. A device used to move along a tabletop in a direction analogous to the direction one desires the cursor to move is called a mouse. A wheel is a rotary device for moving the cursor along a single axis. Timed response refers to collection of the time required to respond. Real-time response refers to collection of responses at several times for a single item. Dynamic responses refers to a continuous analog response. Voice response requires the capability to recognize spoken words.

A keyboard was required by 28 of the 30 item types; only two of the perceptual-motor tests did not require a keyboard. Strictly speaking, a keyboard is not essential ("required") because a touch panel or any of several other modes can be used to perform the same function. However, a keyboard is the most convenient and versatile choice for testing.

A light pen was not required or desirable for any of the test forms; the touch screen is a better option for psychological testing. A touch screen was desirable for 25 of the 30 tests.

Two of the tests required a joystick. It was neither required nor desirable for any of the others. The trackball, the mouse, and the wheel were not specifically required or desirable for any of the tests.

Timed response was required by 16 of the test forms and desirable for 12 more. Real-time response was required by two and desirable for one more. Dynamic response was required by the two perceptual-motor tests and not desirable for any others.

Voice recognition of auditory responses was not required for any of the tests. It was desirable for 10 of them, however.

The questionnaire data suggested that the system should support a simplified keyboard and confirmed the need for a joystick and a touch screen as input options. The questionnaire did not address the need for a timed response, but interviews confirmed a definite need; resolution to a tenth of a second was usually considered adequate. Voice recognition was not seen as a useful option.

Item selection and scoring requirements. Potential users indicated an interest in both mechanical and model-based item selection procedures. Dichotomous, graded, and polychotomous items were all used in sufficient proportions to suggest that the testing models should handle these types. Order-theory item selection was desired, but order-theory scoring received a smaller endorsement. Order-theory testing could probably be disregarded in design of the system. All other forms of scoring including maximum likelihood, Bayesian, and conventional methods were desired.

It appears that the most demanding strategies to be implemented are the maximum-information or Bayesian item-selection strategies coupled with maximum-likelihood or Bayesian scoring. A system capable of handling these methods should be able to accommodate all methods of interest.

Test Administration

Using the 10% cutoff for the questionnaire, as before, several characteristics of the desired system can be enumerated. It should be simple enough to be used by grade school children. Because a proctor will usually be present, the system should be designed to interact with one, ideally by communicating with a proctor's console. The system should monitor all problem conditions that can be reasonably anticipated. The delay between items should be less than one second. The system must be sturdy because it will have to be moved and it may be abused. Finally, both system and media security appear to be essential.

Affordability

The questionnaire data showed a substantial decrease in demand as the system price rose above $3,000. The market was so small above $5,000 that it is probably not worth considering. Thus a target price for the basic system must be $3,000 or less. Obviously expansion options must be available since there will be many functions a basic system cannot support. Such an expanded system should have a target price of under $5,000.

Test Development

The final section suggested that the potential users of the test development system are quite sophisticated in computer use. Therefore, either a menu or an author-language interface for test development can be used and some programming knowledge on the part of the user can be assumed.

II. DESIGN OF A SYSTEM TO MEET THE REQUIREMENTS

The design of a computerized testing system properly begins with a review of existing systems that perform the same or similar functions. The documentation available on current CAT systems was thus reviewed. Unfortunately, no commercial testing systems existed at the time of the review and little documentation was available for the few non-commercial systems that existed. Because computer-assisted-instruction systems bear some similarities to an adaptive testing systems, several of these systems were also reviewed.

Functionally, a computerized adaptive testing system should be able to perform five tasks. Its most basic function is to administer tests. To accomplish that it must also perform the support functions of item banking, test construction, and test analysis and refinement. Finally, it must provide a means of reporting and interpreting the test results. There is considerable latitude in how each of these functions can be accomplished. Following the review of the existing systems, the background and basic design rationale for each of these functions, as designed in the proposed system, is presented below.

A Review of Existing Systems

CAT Systems

The Minnesota system. A large-scale CAT implementation at the University of Minnesota was described by DeWitt and Weiss (1974). The system was developed as a research tool for administering adaptive tests using a variety of strategies. The system consisted of a set of FORTRAN programs developed for a Control Data 6400 that administered adaptive vocabulary and number-series tests. The system, at the time it was described by DeWitt and Weiss, allowed six different testing strategies to be administered. These strategies were coded in FORTRAN and were invoked by entering a seven-character keyword at the beginning of the test. The system started a test session by administering a branched instructional sequence (also coded in FORTRAN) and then it branched to the specified testing strategy.

The Minnesota system was the first major adaptive testing system developed, and as might be expected, it had some shortcomings. The major shortcoming was the limited number of strategies that could be used. This resulted from the fact that the system provided no authoring capability. All strategies had to be coded in FORTRAN. The development of a new strategy thus required the services of a programmer and the implementation of a new strategy, or the modification of an old one, was a major undertaking.

In addition to administering tests, the Minnesota system provided extensive reports of the performance of examinees. While this feature is unusual for a research system, it is essential for an operational testing system. The Minnesota reporting system unfortunately suffered the same shortcoming as the testing system -- the specifications for the reports had to be coded in FORTRAN.

The need for a more efficient method of test specification soon became obvious. When the Minnesota system was expanded and transfered to a minicomputer, a somewhat more user-friendy test specification system was developed (DeWitt, 1976). This system provided a test specification program that allowed a non-programmer to develop a test by choosing strategies and item pools. This program did not offer any features for revising old strategies or specifying new ones, however.

The Civil Service system. One of the most ambitious implementation projects in the area of adaptive testing was a plan to adaptively administer some of the tests of the United States Civil Service Service Commission (McKillop & Urry, 1976). A testing system was developed to accomplish this. This system differed from previous research systems in that it had to be capable of administering only a few tests, all of which used a single strategy (i.e., Owen's Bayesian).

Unfortunately, little more can be said about the Civil Service system. No complete description of the system was ever published and the entire project was abandoned just before its inauguration because of a change in the mission of the Civil Service Commission.

Tne Army system. The Army Research Institute developed a pilot testing system to evaluate the feasibility of CAT (Bayroff, Ross, & Fischl, 1974). This development effort centered on the design and production of a computerized testing station consisting of a CRT terminal, a random-access slide projector, and a response panel. Items were presented on the CRT screen or on a rear-projection screen used with the slide projector. The testing station was interfaced to a Control Data 3300 computer which supported the adaptive strategies. This project placed little emphasis on strategy specification.

The Army system cannot be used for guidance or suggestions regarding the development of a new testing system. Its major emphasis was on the development of a testing station which is now technologically obsolete.

The Missouri system. Reckase (1974) described an adaptive testing system developed to implement a single strategy for administering classroom achievement tests. This system, like most of its contemporaries, had all strategy specifications coded in FORTRAN. The development of a new strategy was thus almost equivalent to the development of a new system. This system offers little insight into the design of a new, more flexible system but exemplifies the systems of its era -- it administered adaptive tests but offered no flexibility in specifying stategies.

TCL. The concept of an author language is not new, having been applied in computer-assisted-instruction systems for years. The first attempt to apply the concept to an adaptive testing system was by Vale (1981). Vale developed a microcomputer-based CAT system primarily as a microcomputer CAT demonstration system. Vale's system was designed to provide (1) a general means of specifying test structures without resorting to a standard programming language; and (2) a minimal delay (at run time) between the response to an item and the display of the next item.

The first of Vale's design objectives was met through the development of an authoring language called Test Control Language (TCL). TCL was a form of programming language tailored to the task of specifying adaptive tests. It included single statements to perform functions that would require many instructions in FORTRAN. Bayesian scoring, for example, was accomplished by a single statement.

Vale's second objective was achieved by compiling the TCL source to a simpler executable form. The textual source instructions were converted to single numbers. Furthermore, complicated statistical searches were converted to table searches at compile time.

TCL also had some shortcomings, however. In TCL, the items of a test were too closely tied to the test itself. For example, certain item characteristics, such as time limits for response to the item, had to be specified in the test itself as part of the test specification. One could argue, however, that such characteristics are dependent on the nature of the item, regardless of which test it appears on. Thus each item should have such characteristics specified at the time of item creation.

TCL also allowed branching to specific items within a test. This can make a test specification difficult to follow and to modify. Whenever an existing item is replace with another, the test developer must then change all branches to and from the item accordingly.

Vale's TCL system provided no special item-banking tools. The standard text editor was used. Each line had a single character code to specify one of four record types. An ideal language would provide a more user-friendly interface.

In general, Vale's system was a successful implementation of CAT on a microcomputer. It provided a more modern system design that was meant for microcomputers, and as such provided a good starting point for the design of a computerized adaptive testing language and the associated CAT software. It lacked, however, an ideal design and adequate documentation for commercial use.

CAI Systems

Computer Assisted Instruction (CAI), or Computer Managed Instruction (CMI), is a field that deals with many of the same issues as CAT. CAI systems must present material to students in much the same way that CAT must present items to the person being tested. CAI also does a certain amount of testing in which actual test questions are asked. It uses information obtained through this testing to determine the speed with which new material can be presented to the student.

CAI systems offer some insights that carry over to CAT. One of the strongest areas of carry-over is the area of authoring languages and procedures. These are the languages and procedures used to build the lessons that are presented to the students.

SILTS. SILTS (SILTS User's Manual), the Scholastic Interpretive Learning/Testing System, is a CAI system used primarily for developing games and simulations. Its authoring system is of interest here primarily because of its complexity of use. The basic unit in SILTS is a "node" which is a file containing statements. Instruction in SILTS is authored by developing nodes using an author language consisting of 37 single-character statements. "P," for example, prints a line of text; "G" causes execution to go to a specific node. An initial node could thus list branches to a string of nodes which could each cause information to be printed.

SILTS is apparently quite a capable system for developing games and simulations. The single-character statements with little mnemonic meaning make it difficult for the occasional author to remain familiar with the language, however. A set of more mnemonic statements would be a definite improvement. Fewer statements might also make it more manageable.

MIL. MIL (Luker, 1979), the Minnesota Instructional Language, is a CAI system with an authoring system implemented as a FORTRAN preprocessor. It is essentially a version of FORTRAN that has been enhanced by the addition of a set of macro-instructions to help the CAI author. Instructions called key-match operators are examples of the macro-instructions. These operators accept a user's response and determine if it matches one or more literal strings. They return a value of "true" or "false" which can be used in a FORTRAN logic expression.

MIL is obviously more efficient than authoring directly in FORTRAN. FORTRAN, however, is not an acceptable authoring language because it is too difficult to learn. MIL is even more difficult to learn because it requires a knowledge of FORTRAN as a precursor. It thus does not succeed in allowing test authors to avoid learning a programming language.

GENIS I. Bell and Howell offer a commercial CAI package called GENIS I (Bell and Howell, 1979). It is composed of two separate CAI systems. CDS I, the Courseware Development System is the higher-level system. It consists of a menu-driven authoring subsystem and a presentation subsystem. The menu-driven aspect of CDS I is fairly simple. It is essentially a fixed sequence of questions aimed at collecting the required information. This repetitive and lengthy process may be unappealing to an experienced author, however.

CDS I provides some simple formatting of textual items. This can be important to CAI systems that must present large quantities of text. It may be less useful for CAT whose major emphasis is testing rather than instruction. Free-response answers can also be handled by CDS I, though in a limited fashion. It allows the test author to specify groups of correct answers, with certain allowable variance in the spelling. That leniency in the spelling, however, is limited to the explicit specification by the author of the alternatives, including the use of a wild-card option on selected letters. This is a process that might better be made automatic, through the use of any of several current algorithms.

The other half of the GENIS I system is the MARK-PILOT CAI system. MARK-PILOT uses an author-language approach to CAI. It is based on the PILOT language for CAI with some extensions for Apple Graphics. CDS I routines can call MARK-PILOT routines but not vice versa. PILOT, like MIL and SILTS, suffers from being too simple in design and too complicated to use. Single-letter keywords make it difficult to read. Further, it fails to introduce any high-level concepts meant for dealing with CAI.

PLATO. One of the largest and most well-known CAI systems is Control Data Corporation's PLATO system (Control Data, 1978a, 1978b). PLATO is an extensive system that allows test items and learning material to be presented in a variety of media. One terminal used by PLATO, for example, incorporates a touchscreen, a microfiche display, and extensive graphics capability.

PLATO allows authoring at two levels. An author language level allows details of graphic displays and complex branching sequences to be specified. It is a very extensive language; it is thus very powerful and requires a good deal of experience to become familiar with it. The second level of authoring is through the CMI system. This is a menu-driven system that allows subject-matter experts to develop instruction or tests without learning the author language. The CMI system is much easier to master than is the author language.

The PLATO system is an elegant CAI system, if only for its completeness and capabilities. For the current design, however, it is its dual-level authoring capabilities that are most interesting. Ideally, a CAT system would have a similar system. The author language would be used to lay out the strategies. The menu system would then be available to select the items to be included.

Directions Suggested by Current and Past Systems

Two conclusions regarding the state of the art can be drawn from the review presented. The first is that, until recently, little emphasis has been placed on the development of a general-purpose CAT system for developing and administering tests using a variety of stategies. None of the systems allowed a test developer with minimal computer knowledge to develop or modify an adaptive testing strategy. Vale's system represents a step in the right direction but the sophisticated authoring-language systems used for CAI suggest that there is much room for improvement.

The second conclusion, drawing heavily upon the PLATO design, is that several levels of author interfacing may be useful. Testing strategies may be impossible to specify completely without a complex author language using concepts similar to those found in programming languages. Such complex author languages may be too difficult for certain users to master, however. A simpler, but somewhat less flexible, menu-driven system should be supplied for use by these individuals.

Test Construction

The test construction system is used to create a test to be administered by the test administration system. It draws specific items from the item banking system, incorporates them into the testing strategy, and selects the information that should be recorded for test analysis and interpretation. As was suggested by the questionnaire analyses and literature review, the proposed test construction system includes facilities for constructing tests either directly through an author language or indirectly using a menu system to generate the author language.

Author Language

Tests may be specified directly using an author language called CATL (Computerized Adaptive Testing Language). CATL is a language explicitly designed for specifying adaptive tests.

Throughout the CATL language, mnemonic simplicity has been used to guide the choice of statement names, labels, and variables. Statement names were chosen to describe the functions they perform. Label and variable names in CATL can be long alphanumeric strings as opposed to the single-character or numeric names used in some languages. This allows test developers to use descriptive names, which makes tests easier to understand. Furthermore, the language was developed to allow test specifications to be written in a modular style.

A special line-termination convention was used throughout CATL. Many modern programming languages ignore line structure altogether so that statements may continue from one line to the next without a line-continuation character. Although this makes the language more flexible, it requires every statement to end with a statement-termination character. In Pascal, for example, misplaced statement termination characters are the most common cause of program errors. For this reason, a very simple line continuation was chosen for CATL: every statement ends at the end of the line unless it is continued on the next line with an _&_ character. This is especially well adapted to CATL tests since few CATL statements require more than one line.

CATL consists of seven distinct groups of executable statements and an additional set of statements used by a menu processor. These are described below. A more complete description with examples is provided in Chapter 8 of the the draft User's Manual in Appendix A.

Module delimiters. Myers (1975) discusses "structured programming" and "modularity" as ways to divide large programs into smaller programs that exchange a minimum amount of information. Small independent program modules are easier to understand than are large complex programs, but there must be mechanisms to (1) define independent modules, and (2) pass information between modules. In CATL, the TEST and ENDTEST statements are used as module delimiters.

Information communication within and among subtests is done through variables. CATL provides two kinds of variables. Local variables can be used within a test (after a TEST statement and before the matching ENDTEST statement) without conflicting with the variables in any of the other tests. Global variables can be used for information common to all the tests, or to pass information between tests. Because of the many built-in adaptive testing features in CATL, few variables should be needed in CATL programs. This being the case, using global variables has advantages over passing arguments from module to module; argument passing is fairly complicated and is often a source of programming errors. Global variables are declared in CATL by preceding the variable name with a _@_ character.

Assignment statement. The second group consists of a single assignment statement, SET. SET is used to set variables, either local or global, to expressions including other variables and/or constants. In CATL, assignment is a minor feature.

Basic statements. The third group includes the basic statements for inserting comments and for presenting an item or group of items. An essential part of any programming language is a mechanism for inserting comments into a program to document its design. The _!_ character is used in CATL to mark the beginning of a comment because its vertical nature graphically separates the program from the comments. CATL allows any number of characters following an _!_ to be used for comments. As in most languages, a CATL comment (preceded by the comment delimiter) may fill an entire line; unlike many languages' comments, however, a CATL comment and its delimiter may follow other statements on the same line. The end of a line terminates a comment. PL/I, Pascal, and several other languages require both the beginning and the end of a comment to be marked with a special character. However, if an end-of-comment marker is accidentally omitted, the instructions following the comment will be ignored by the compiler. Allowing the end of a line to terminate a comment eliminates the possibility of this error occurring.

In CATL the administration of an item is the fundamental operation. The administration of an item is specified by putting the item identifier on a line and preceding it by the # symbol.

Item development and test development are separated to allow a single pool of items to be used in many different tests; items are not modified at the time a test is constructed. However, item characteristics included in an item may be overridden when the item is used in a CATL test. Removing the specification of items and item characteristics from the test-construction process simplifies the tests; allowing item characteristics to be overridden provides flexibility for special applications.

It is often desirable to develop tests or item pools that can be included in several different tests. An auxilliary include operation (not actually part of CATL) is provided to accomplish this. The * symbol, followed by a file name, allows information from another file to be copied into the test. This statement avoids the need to keep copies of the same information in more than one test.

Testing-strategy statements. The CATL author language uses several building blocks or strategy primitives to specify strategies. Complex testing strategies can be constructed by combining these primitives with scoring algorithms and conditional logic. The first primitive is post-item branching. In a CATL test with no branching, administration of each item in the test proceeds sequentially until the last item (or line) in the test is reached. Branching allows administration to continue at a different line in the test, depending on how the examinee answers any of the previous items. Specifically, CATL permits branching to a different line in the test depending on (1) whether an item is answered correctly or incorrectly, or (2) which one of several possible responses to the item is chosen.

The other two strategy primitives use pools of items. The SEQUENCE statement allows individual items to be grouped together as if they were logically one item. Each time the SEQUENCE statement is executed, the next item (starting from the one at the top of the sequence) is administered. The SEQUENCE is terminated by an ENDSEQUENCE statement. Item characteristics may be overridden by items within a SEQUENCE statement in the same way that they are overridden in a # statement. The SEQUENCE statement itself may contain only branching information.

Although the SEQUENCE statement is general enough to have several uses, it is especially useful for implementing inter-subtest branching strategies. In these strategies, branching is based on an item response.

CATL uses the same syntax for branching on the SEQUENCE statement as it does to branch to individual items. The implementation of branching strategies in other testing languages such as TCL (Vale, 1981) required pointers to the items in a test. Pointers within a program are very difficult to implement, and as a result they have been avoided in modern general-purpose languages. CATL's SEQUENCE statement automatically keeps track of which items in each SEQUENCE statement have been administered, so no pointers are needed.

Several different conditions resulting from the administration of an item (including "correct" or "incorrect" responses) can cause branching to a different line in the test. Like most modern programming languages, CATL allows branching only to lines marked with a program label. Program labels always begin with a $. Branching to labels outside the test is not allowed.

The SEARCH statement is another specialized adaptive testing statement provided for model-based testing strategies. The SEARCH statement is similar to the SEQUENCE statement except that instead of administering the items in sequential order, it administers the "best" remaining item in the pool. The best item is the one that provides the most psychometric information for a given ability level. The SEARCH pool is delimited by SEARCH and ENDSEARCH statements. Although the model-based strategies require a great deal of computation, almost all of it is done prior to test administration. Different model-based strategies may be implemented by searching on different scores.

Conditional statements. The fifth group of statements includes the conditional statements. Three statements -- IF, ELSEIF, and ENDIF -- constitute the conditional group. IF causes execution to skip to the next ELSEIF or ENDIF unless the specified logical expression included in the statement is satisfied. ELSEIF functions similarly but must be embedded in an IF statement. Both IF and ELSEIF branch to the ENDIF after successful execution of the clause.

Early programming languages used an unstructured IF-GOTO construct. As languages evolved, the IF-GOTO construct was improved to an IF-THEN-ELSE construct. The IF-THEN-ELSE is sufficient for structured programming when used with block delimiters (such as the BEGIN-END in Pascal). Other languages have adopted an ELSEIF construct to allow for multiple conditions. CATL combines the advantages of all these constructs and removes the need for block delimiters with an IF-ELSEIF-ENDIF construct. Any number of ELSEIFs may be included in the construct. If no conditional logic follows the ELSEIF, it functions like the unconditional ELSE in some languages. An explicit ELSE was not used in CATL so that the number of system keywords could be kept to a minimum.

Declarative statements. The declarative group of statements sets up conditions for a test module. The SETSCORE statement identifies the scoring routines that will be used and the variables that will be used for the scores. The variables are automatically updated with values from the scoring routine whenever they are used. In a general-purpose language, the scoring routine would have to be called explicitly each time the variable was used, requiring many extra program lines. This has been avoided in the design of CATL by making the CATL system responsible for determining when to update the scores. Thus, the CATL system keeps track of when scores are to be updated and updates them only when they are used. Scores are not unconditionally updated after every response.

To provide an easy mechanism for ending tests when the given conditions are met, CATL uses the TERMINATE statement. This statement was intended primarily for tests using model-based branching strategies that administer items until the estimate of the examinee's ability becomes sufficiently precise. The TERMINATE statement may be used in conjunction with any other testing strategy, however. This makes the specification of termination conditions more flexible than it would be if it were built directly into the SEARCH or SEQUENCE commands. The design of the TERMINATE statement is consistent with structured programming doctrines. Although this type of statement is uncommon in programming languages, it is similar to the structured BREAK statement in the C programming language.

Output control statements. The output control statements provide a mechanism to permanently store test scores and other data. KEEP is very similar to the WRITE statement in languages like FORTRAN. In some testing applications, the values of several variables will need to be stored after each item is administered. To simplify these applications, the AUTOKEEP statement may be used to automatically store these variables after every item.

Menu System

All of the statements described above are part of the CATL author language. Author languages provide much more flexibility than menu systems, but a test developer must study the author language before a test can be written. CATL supports a menu system that allows people with very little programming experience to develop tests. Instead of developing an inflexible menu system, CATL menus are actual CATL tests with "blanks" to be filled in by the menu system. The final group of statements, the menu statements, allow control of the menu system. The INSTRUCT statement in a CATL test is read by the menu system, and the instructions following the statement are displayed for the test developer. The developer enters his/her responses to the instructions into the CATL program at the specially marked places. This unique approach combines the advantages of a menu system with the advantages of an author language.

Item Banking

The heart of any testing system is the set of available items. The process of entering, maintaining, and ordering these items is known as item banking. First, an item must be created: It must be named and described, and the body of its text must be entered into the storage system of the computer. After it has been created, it may still undergo changes. The author of the item may wish to alter the wording of the text, change part of the item's description (e.g, parameters) or otherwise modify the item. These alterations should be made without the need to re-create the entire item. Making the creation and editing processes simple and similar to each other is an advantage for the user.

Finally, the item banking system must be able to support specialized items. Such special items allow the author to make use of sophisticated options for a CAT system (e.g., videodisc displays and graphics). Here again, item banking deals with the entry and editing of such special items.

Item Classification and Storage

Underlying the entry and editing process for individual items is the file system that supports the organization of the items into useful collections or pools. Items are typically grouped together according to some criterion, usually pertaining to the subject matter of the item. Without such grouping, it would be difficult to manage the many items created by test developers.

Each item needs a unique identifier. In most item banking systems this consists of a content-area identifier and a unique item number within the area. Previous banking systems have used content identifiers based on mnemonic codes, Dewey decimal codes, and simple descriptive codes. For banks of small to moderate size, a simple descriptive code is probably best. For example, such a code might be six letters long. The item identifier might then be six letters followed by a number for identification within that area.

Closely related to the classification and naming of items are the mechanisms for item storage and retrieval. The item name provides the first link in the search and retrieval of an item. An on-line directory of item names and storage location addresses typically provides pointers for use by programs that need access to the items. It is used in much the same manner as a telephone directory. For small systems, large master directories are not always feasible, however. Limited disc storage space makes it impossible to maintain lists of thousands of item names and corresponding disc addresses.

The storage technique that was developed for the proposed CAT system attempts to make as much use as possible of the underlying operating system's file storage facilities. Directories of files are maintained by the operating system for the organization and access of data files. When applied to item banking, these directories can be used as the first level of organization of an item banking scheme.

To take advantage of the operating system, then, the name of the item must contain, in part, some information pertaining to its location. This can be readily accomplished by implementing a scheme that uses the first n characters of the item name as the filename of the pool in which this item is stored (where n is less than or equal to the maximum allowable filename size for the given system). For most systems six characters are allowable. Very large item banks or very complex naming schemes may require more complex software algorithms for efficient storage and retrieval. For the proposed system, the file name approach should suffice.

The remaining characters of an item name are then used to locate an item within the pool. This is accomplished by using the directory that the item bank maintains at the beginning of each file. For example, to find the item named VOCABU001, the first six letters are used for the filename (thus the file "VOCABU" is opened) and within that file, its directory contains the remainder of the name, "001."

Items themselves are stored with both header and body together as a single record. The header contains the various item characteristic parameters and other flags that indicate the type of the item and its various execution requirements. The body of an item contains its textual information. Since the body of an item can be of any length, the item records that make up a file of items must be variable length records.

Special Items

Videodisc and graphics displays were the two most desired display options for the CAT system prototype. To support these options, the item banking system must include in its design features that facilitate entry and editing of these options.

Videodisc. The videodisc option to the proposed system will be implemented using a standard serial interface to the videodisc and an auxiliary TV monitor beside the video screen/keyboard assembly. Videodisc pictures will be displayed on the auxiliary monitor while any explanatory text will appear on the CAT video screen. More extensive features for combining videodisc and computer displays were considered to be beyond the scope and price constraints of this system because of the technical complexities involved in mixing computer and videodisc signals on a single screen (see Bejar, 1982).

Three basic functions are supported with videodisc: (1) displaying a single frame, (2) displaying a range of frames, and (3) looping through a range of frames repeatedly. If an item uses the videodisc, three additional fields are used in an item's header. These fields are: starting frame number, final frame number, repeat flag. If the starting frame is zero, then this item does not use the videodisc option. For displaying a single frame, the start and end frame numbers are identical. The repeat flag, if TRUE, specifies that the range of frames from start to end should be displayed over and over until the response time expires or the question is answered.

Graphics. Graphics are stored in the body of an item. The header must contain a flag that, if set, indicates that the body of the item contains graphics.

If point graphics are to be used, the item author needs the digitizer option for the system when the item is being created or edited. (Systems for administration of graphics items need not be equipped with such a digitizer.) One form of digitizer consists of a specially wired pen and pad arrangement that allows the user to move the pen across the pad's surface. The x-y position of the pen is then sent to the CPU. Special programs for the entry and editing of graphics with the digitizer are readily available. Such a program will be used for graphics editing on the proposed system and its output will be entered as the body of a graphic item.

Alternatively, graphic items can be stored as control instructions for a graphic terminal. In this case, graphics items can be stored as textual items. No graphics system software needs to be developed for this mode of entry.

Creating and Editing Item Text

The interface most frequently used in item banking performs the functions of entering and editing an item. It should be as user-friendly as possible. The most efficient and user-friendly editors are screen-oriented editors. A screen editor displays a portion of a text document and allows the user to make changes in the text that are immediately apparent on the CRT screen. Since the CAT system prototype will have a CRT screen, the use of the screen's features for editing should play an important part of the design of the editor.

A combination of screen-oriented approaches was taken in the proposed system. First the item header information (inserted into a number of small data fields) is requested in a fill-in-the-blank mode. The author can see what information is being requested and can fill in any or all of the appropriate information. The fill-in approach seemed most appropriate for headers since they are generally of a fixed format. This fill-in-the-blank approach has been used in systems such as Plato (Control Data, 1978a, 1978b). The question-answer approach has been used by some (e.g., GENIS I by Bell & Howell, 1979) but this is often too slow for the experienced user. The fill-in approach displays almost all the various fields at once, allowing the author to see what is required, giving the whole picture and thus making the system easier to use.

Once the header has been completed to the author's satisfaction, editing shifts to a free-format approach for the body of the item. Here the screen-oriented editing features refered to above are used.

Test Administration

The test administration system needed by most users must perform four functions:

1. It must instruct examinees how to use the testing system and, in general, how to respond to items.

2. It must display the items to the examinee.

3. It must accept responses to the items, edit them for proper format, and score them.

4. It must pass information to a monitoring system that can keep a proctor informed of each examinee's status.

Instructions

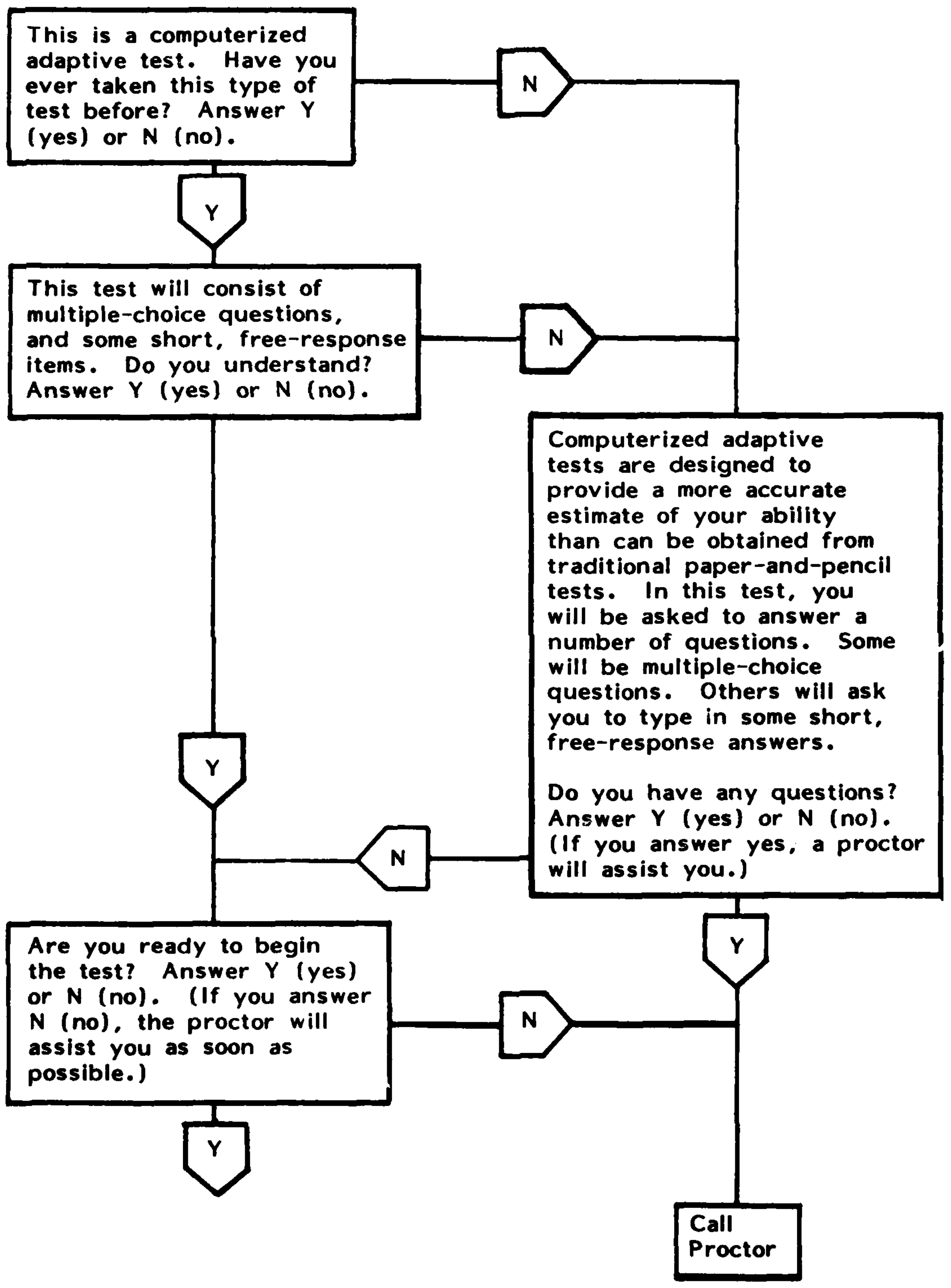
A basic part of test administration is teaching the examinee how to respond to the test. In paper-and-pencil testing, the instructions are read by a test proctor as the examinee follows along in the test booklet. A computerized testing system can free the proctor from this duty and interactively administer instructions to each examinee, providing clarification where it is required. As DeWitt and Weiss (1974) pointed out, computerized instruction must be capable of branching on the basis of examinee responses. By branching on a response, the system can produce instructions of the type shown in Figure 3. Examinees who have previously taken computerized tests or who read and comprehend the instructions quickly can begin sooner than other examinees. Those who require more extensive explanation receive the specific instructions they need to start the test.

Branched instruction is, in a sense, simply a form of adaptive testing in which the instructional screens are treated like (unscored) test items. A computerized testing system capable of branched adaptive testing can thus perform the instructional sequence as a type of test at the beginning of the session. The proposed system is designed to handle instructional sequences in this manner. Instructional screens will be entered as items and the instructional sequence will be specified as a branched adaptive test.

Item Presentation

Item presentation is the central task in test administration. Each test item must be displayed for the examinee. This is typically done through

Figure 3. Sample Instructional Sequence


This is a computerized adaptive test. Have you ever taken this type of test before? Answer Y (yes) or N (no).
N
Y
This test will consist of multiple-choice questions, and some short, free-response items. Do you understand? Answer Y (yes) or N (no).
N
Computerized adaptive tests are designed to provide a more accurate estimate of your ability than can be obtained from traditional paper-and-pencil tests. In this test, you will be asked to answer a number of questions. Some will be multiple-choice questions. Others will ask you to type in some short, free-response answers.
Do you have any questions? Answer Y (yes) or N (no). (If you answer yes, a proctor will assist you.)
Y
N
Are you ready to begin the test? Answer Y (yes) or N (no). (If you answer N (no), the proctor will assist you as soon as possible.)
Y
N
Y
Call Proctor

audio or visual modes, visual being the more common of the two. In designing an item display, both the display information and item structure must be considered.

Display information. In a textual item, the display information consists of characters. In a graphic item, the information may consist of graphic characters, characters controlling a graphic terminal, or point representations of the graphic display. Obviously a means must be provided for transferring this information to the display device.

In addition to information transfer, some control functions must be passed to the display device. The most basic of these is the "clear screen" function. A function is also needed to control the timing of a screen display. In simple items, this function needs only to control the length of time the item is displayed on the screen before the screen is cleared. Additional control functions may be needed to move the cursor to particular locations.

In the proposed system, item information will be entered using a special-purpose item editor. This editor will be designed to communicate with a graphic digitizer for graphic input. It will also set explicit variables to perform the necessary control functions.

Structure. Presentation structure is defined by the logical categorization of item components. For example, a basic multiple-choice item consists of a stem that asks a question and several alternative responses, only one of which is correct. Typically, this can be considered one parcel of information to be displayed.

A more complicated item may require a premessage or a postmessage. A premessage may be used to warn the examinee that an item stem is about to be presented. Such an item may require, structurally, several premessages. The first would be the warning, the second might be a blank screen between the warning and the item stem (to provide a pause to clear the examinee's visual memory, for example), the third premessage might be the actual item stem, and the fourth might be another pause. The alternatives would constitute the entire item and would be presented on a separate screen display.

The purpose of a premessage is to allow several portions of an item to be presented in a precisely timed manner. Each of the messages could logically be considered an item. However, if each message required a disc access, the timing between the messages would not be precise. The premessage structure allows all portions to be gathered into computer memory before presentation of the sequence is begun.

A postmessage is used to append additional information onto the end of an item. An example of where a postmessage might be useful is for adding the message "Don't forget to press the return key" to the end of the item. A postmessage can also be useful for providing feedback on the response given. To do this, however, the postmessage must be conditional on the response given.

The proposed system is designed with a structure containing both premessages and postmessages. The premessages may be stacked so that several may be administered. This means that premessages may refer to other premessages and that the first one administered will be the last one referenced. Similarly, the postmessages may be chained. This means that a postmessage may refer to other postmessages and they will be presented in the order that they are referenced. Postmessages cannot be conditional on the response in the proposed system, however, only the item itself can accept responses. Premessages and postmessages simply present information to the examinee.

Response Acceptance and Editing

Error correction. Accepting and editing item responses is the next task for a computerized adaptive testing system to consider. It is unrealistic to expect that all examinees will enter their responses perfectly. This raises the issue of how to allow examinees to correct their errors.

Short responses will probably be solicited for most testing applications. Therefore, the easiest and most efficient means of correction is probably a destructive-backspace key. This key permits correction of the most recent character entered and can also erase an entire entry if necessary. A non-destructive-backspace key permits characters to be inserted in a previously typed string of characters and is useful for correcting long responses. It may be cumbersome and difficult to use, however. The proposed system will thus have destructive-backspace error correction.

Special responses. Provisions should be made to allow examinees to request assistance, omit items, or to otherwise interrupt the regular sequence of item presentation for any reason. Ideally, these special responses should each be contained on a single key. A long list of such responses could be implemented, but the most important are HELP and PASS.

HELP should be designed to be accepted at any point in a testing session. It will direct the administration system to freeze the test and signal the proctor. The signal will continue until the proctor responds. Should an examinee choose not to respond to an item, the PASS key would enable him/her to skip that item quickly and easily.

The proposed system will support HELP and PASS function keys.

Timing. Provision should also be made for controlling the length of time during which a response will be accepted. For some items, this time can be controlled by the amount of time the item display is left on the screen. For others, such as items displayed for a fraction of a second, a separate response time should be provided. In the proposed system, stimulus and response times will be contained as characteristics of the items.

Examinee Monitoring

Even though the system software is designed to be as user-friendly as possible, it may still be necessary to provide personal supervision and assistance. An examinee-monitoring system (cf. Prestwood, 1980) would allow this. This assumes there will always be a proctor present at the monitoring station; responses to the questionnaire suggested that this assumption is valid.

The proctor will, at a minimum, start the test from either the monitoring station or the examinee's terminal. The proctor should also be able to stop or interrupt a test (pause) without losing the response data accumulated to that point.

As mentioned in the previous section, an examinee could signal the proctor for assistance with the HELP key at any time. If a proctor's terminal was not available, the examinee's terminal could beep or flash. If a proctor's terminal was available, the help-needed indicator could flash at the monitoring station until the proctor responded. A proctor's terminal could also be used to track the examinee's performance. This would allow the proctor to determine if the examinee needed help and would allow the proctor to see where in the testing sequence an examinee was.

Regardless of whether a proctor's terminal was available, the monitoring system could constantly evaluate the examinee's response pattern to detect problems such as "sleeping", random responding, and coached responding (Prestwood, 1980). Algorithms are available for detecting some of these problems and should be useful in informing the proctor of problems s/he might otherwise fail to detect, especially when large numbers of examinees are being tested.

The basic concerns of the system design with regard to the monitoring system are the design or selection of the error-detection algorithms and the provision of communication between the testing stations and the proctor's station. The former has been provided elsewhere (cf. Prestwood, 1980) and the latter is dependent on the hardware configuration. The proposed system will provide capability for error-detecting algorithms and will, as an option, support the proctor's console.

Test Interpretation

If the system is to be easy to use, score interpretations and displays must be easy to produce. An interpretation should be an easy-to-read, well-formatted presentation of the test results. Interpretation is not a trivial task. The user must be able to specify detailed instructions for the interpretive function to use. However, these instructions should be easy for the user to understand and/or specify.

At one extreme, the specifications could be quite simple. The data would be stored (for both pre-defined or user-defined tests) in a standard format and the user would not need to know anything about the internal software mechanisms. The results, in this case, would be displayed in a uniform manner. The obvious drawback to this scheme is the uniformity it imposes. Uniformity is not necessarily undesirable, but in this case it reduces the flexibility of the overall system. Care was taken to allow the user to design custom testing strategies. To force these strategies to use a common interpretation would be a mistake.

At the other extreme, an interpretive system could be devised that would allow the user to completely control the interpretation and display of test results. While such a system would permit maximum user control, it would also require the user to be well versed in system details concerning the storage, retrieval, and display of data. This would be an unnecessary burden on the majority of system users.

The compromise selected for the proposed system allows users to specify any number of different types of interpretations using a structured author language. The language employs a modular design. Modules contain textual materials ranging from cursory descriptive phrases such as "High Ability" to complete narrative descriptions in paragraph form. Each module also contains a logical expression that determines whether or not the textual material is to be output for a particular individual. The logical statements are similar in form to logical expressions in FORTRAN or CATL.

For instance, if the description "High Ability" is to be printed whenever an individual's Bayesian modal score exceeds 2.0, the module below is appropriate.

```
[#ABL001
(BSCORE > 2.0)
High Ability
#ABL002]
```

In this example the module's name is #ABL001, the first name after the opening brackets. If BSCORE exceeds 2.0, the text High Ability is printed. The last line of the module calls another module, #ABL002, which may present additional information. That message, as well as the one in the module above, could have been several paragraphs in length.

Modules can also cause an individual's actual scores to be printed when that is desired. These scores may be embedded in text to provide labeling for them.

Test Analysis

Two types of analyses are likely to be useful in a CAT system: conventional item and test analyses and IRT item and test analyses. The conventional analyses consist of estimates of a test's reliability, the proportion of examinees answering each item correctly, and correlations between the item responses and the total score. IRT analyses primarily include estimation of the three IRT item parameters, item and test information, and model-data fit.

Conventional Analyses

Conventional analyses are computed at the item and the test levels. At the item level, two basic statistics are usually computed: item difficulty and item-total-score correlation. The conventional item-difficulty statistic is the proportion of individuals in a sample that answer the item correctly. This requires very little computation. Similarly, the item-total score correlations can be computed for a large number of items in a test with little disc storage required.

At the test level are test reliability and distributional statistics. The reliability estimate usually calculated in a conventional item analysis program is a Kuder-Richardson Formula 20 (KR-20; cf. Nunnally, 1967). The computations required are not computationally taxing nor do they require extensive memory. They can be easily implemented on almost any small microcomputer. The distributional statistics are usually limited to the mean and standard deviation of the score distribution, although the skew and kurtosis are also sometimes computed. Additionally, the standard error of measurement, a function of the reliability and of the standard deviation of scores, is often calculated.

The following conventional statistics were included in the design of both the basic and the extended CAT systems:

1. Proportion of correct endorsements for each item
2. Correlation between each item score and the total score
3. KR-20 test reliability
4. Score mean
5. Score standard deviation
6. Standard error of measurement

IRT Analyses

While conventional item analyses are helpful in refining items, IRT item parameters are essential in an adaptive testing system. The IRT item parameters, in general, describe the item's difficulty, its power to discriminate among ability levels, and its susceptibility to guessing. Although several IRT models are currently available, the most general one in popular use is the three-parameter logistic model. In this model, $\underline{a}$ indicates the item's discriminating power, $\underline{b}$ its difficulty, and $\underline{c}$ the probability that an examinee with very low ability would answer it correctly.

Unlike the conventional analyses, IRT item analyses are computationally burdensome and may require a good deal of computer memory. There are several methods for estimating these item parameters including (1) a set of transformations from the conventional item statistics, (2) a method that minimizes the lack of fit to the IRT model, and (3) a maximum-likelihood approach that maximizes the joint likelihood of a set of item and ability parameter estimates given the observed data. The transformational approach is computationally the least taxing but does not yield particularly good estimates unless a fairly strict set of assumptions is met. The maximum-likelihood approach is, theoretically, the best of the three approaches but requires much computation and a large amount of memory for intermediate storage.

IRT test analyses are, in some ways, functionally similar to the conventional analyses. The primary IRT test statistic is the test information function that is proportional to the inverse of the squared conditional standard error of measurement. From this function, the test reliability can be estimated for any specified population of examinees. Alternatively, the information function can be averaged over the examinees in the population. Computation of the information function is relatively simple once the item parameters have been estimated. Other test statistics of possible interest include the means of the item parameters and their standard deviations.

The IRT item analysis capability will be available on the extended system. The method of analysis will be the method of maximum likelihood. This calibration capability will be available for dichotomous items using the three-parameter logistic model. The proposed basic system will not contain any IRT item analysis capabilities because it will not have sufficient computing power nor will it have disc storage with sufficient capacity or access speed.

III. DESIGN OF A SOFTWARE PACKAGE TO IMPLEMENT THE SYSTEM

The proposed adaptive testing system is composed of programs and data files. These are shown grouped into subsystems in Figure 4. The programs perform the basic functions of the adaptive testing system and are grouped into five subsystems: Test Construction, Item Banking, Test Administration, Test Interpretation, and Test Analysis. The data files provide a means of communication between the programs. Although the data files are shown within subsystem boundaries in the figure, the data files are typically typically shared by two or more of the subsystems. In addition to the five subsystems, Figure 4 also shows a system editor. This editor is one of the operating system programs. Although not shown explicitly in Figure 4, the operating system is also an essential part of the proposed system. Each of the subsystems is described in some detail below.

Test Construction

Template Processing

Templates are test-specification shells written in CATL. They differ from complete test specifications in that they contain holes marked by special placeholders indicating the positions for item identifiers and other pieces of information. A template can be "filled in" by the test developer to produce a complete test specification. This test can, in turn, be consolidated into an executable test file and then administered. This filling in, or preprocessing, of templates allows a novice user to construct tests easily. It also provides a major convenience for those test developers who need to construct a variety of tests that use the same test strategy. For more information concerning the creation of tests using templates, as well as information on the templates themselves, see Chapter 5 and section 8.9 of the draft User's Manual (Appendix A).

The design of the template processor is shown in Figure 5. It depicts the five major functions of the preprocessor as a hierarchy. The functions in this hierarchy, which is a system design document, can be translated into procedures that operate as described below:

Input and output. The preprocessor, called Fill-Template in Figure 5, reads through the test-template (see Figure 4) using the procedure Read-Source. Read-Source examines each line to see if it contains any of the special template constructs (i.e., INSTRUCT statements or blanks to fill in). If no menu statements are found, the line is written in its original form to the test-specification file (Figure 4) via the procedure Write-Test.

Figure 4. System Overview

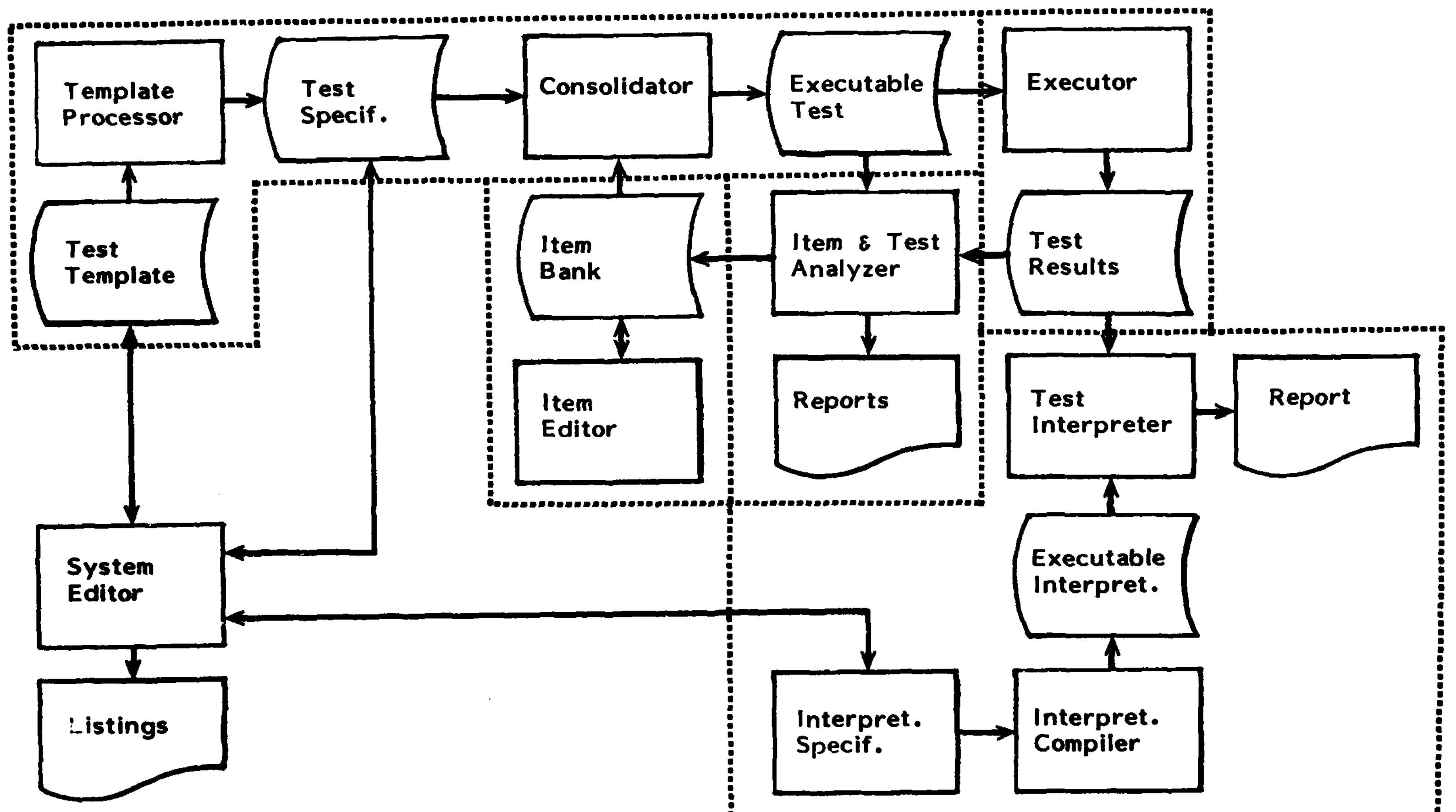
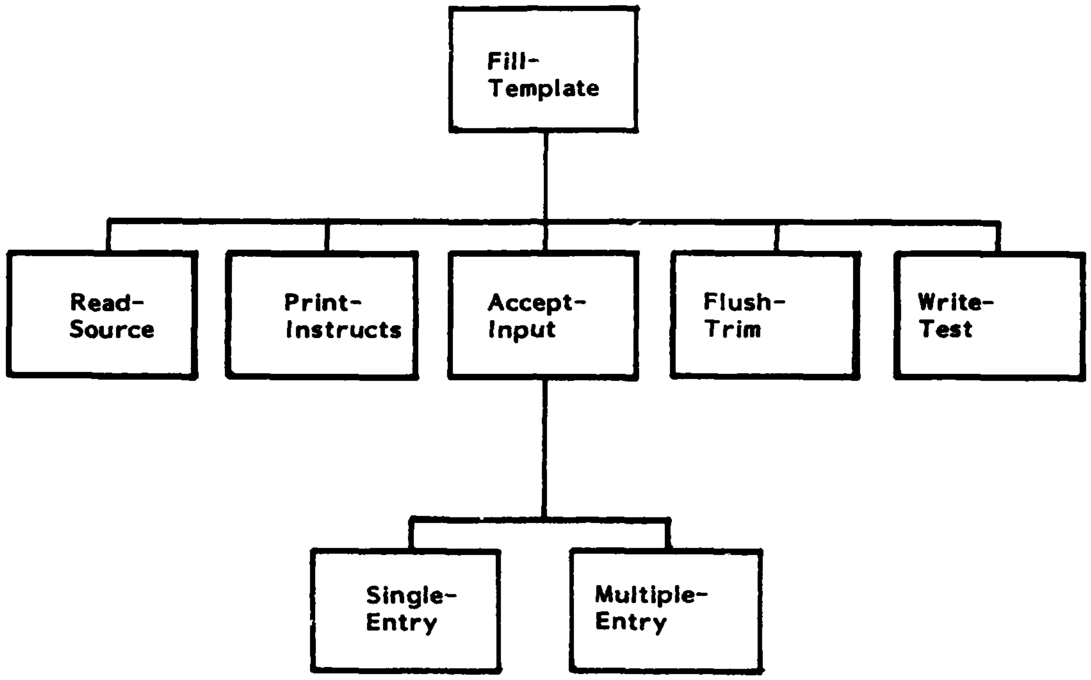


Figure 5. Template Processor Overview



Print-Instructs. If an INSTRUCT statement is encountered at the beginning of a line, then the remainder of the characters following the keyword INSTRUCT are displayed on the user's screen. This displayed text contains instructions from the template author to the test developer concerning the type of information to be supplied, the format that is required, etc. The INSTRUCT statements are not copied into the test file. Furthermore, they cannot be continued onto a second line. Multiple lines of instruction require multiple INSTRUCT lines.

Accept-Input. The presence of an underline character ("_") indicates that the user needs to supply information, thereby "filling in the blanks" of a template. A # character displayed after an underline indicates that multiple, repetitive input is required. This aspect of the system design permits a user to easily input item lists of unspecified length.

Both types of input (single- and multiple-entry) are handled by the Accept-Input procedure. Accept-Input deals with these two cases by invoking one of two separate procedures, whichever is appropriate. For the simple case, the Single-Entry procedure accepts input from the test developer; this input is used to replace the underline character directly.

Typically, item identifiers will be input to the preprocessor, although the system does not restrict the use of underlines to item identifiers. Any type of input will be accepted, inserted at that point, and written to the test file. It is up to the template author to provide sufficient INSTRUCT statements to explain the input requirements.

The procedure Multiple-Entry is called to handle the repetitive case. Here again the user's input replaces the underline characters, and the new line is written to the testfile. However, in contrast to Single-Entry, the preprocessor does not move on to the next line in the template file. Instead, the same line is repeated: New input is requested from the user and, if supplied, replaces the underlines. This new line is written to the test file. This process continues until the user responds to the input with a special function key to terminate it.

Flush-Trim. To allow additional control and thus enable a single test template to be used for tests of slightly different structures, another special function key is provided. This key directs the preprocessor to finish the process of building the test without any more input. When this key is pressed in response to a user-input request, the preprocessor sets a flag to indicate that it is in "flush" mode. When in flush mode, the Print-Instruct and Accept-Input programs are no longer called. Instead, a call is made to the Flush-Trim program. This program operates on each line, trimming off all but the label field, and returns the modified line to be written to the test file. (The labels must be preserved so that branches can be made from previous statements to that point.)

An example of how this capability might be used can be seen in the development of a template for a stradaptive test. The template might be developed to handle a maximum of, say, 15 strata. Each stratum requires several CATL statements to set it up and to distinguish it from other strata so the multiple-entry feature (which has its own termination capability) cannot be used across strata. However, with the Flush-Trim procedure, a user can terminate stratum completion after any number of strata. A single template is thus useful for several different forms of a stradaptive test.

Summary. In the test construction subsystem, a CATL template is itself a CATL program with INSTRUCT and other special statements embedded in it for use by the menu processor. Through interaction with the user, the menu processor produces a standard CATL program by copying the CATL template and replacing the INSTRUCT and underline statements with other statements. Template processing, when complete, produces a test file of CATL statements. This file is then given as input to the Test Consolidator which converts it into an executable test. Template preprocessing provides a simple set of constructs for use by the template author, thereby making the authoring task as clear and easy as possible.

Test Consolidation

Test consolidation is a process very similar to program compilation that is used to minimize the processing required at run time. In the consolidation process, special processing is performed on SEARCH lists to produce a table for item selection at run time, item identifiers are converted to item addresses, and the CATL source statements are converted into a shorthand form that can be quickly processed and is more compact than the source test specification. Once test consolidation is complete, the test is ready for administration. As such it can, if desired, be transferred to a smaller system for administration.

Consolidation is a three-stage process. The first stage converts the CATL source statements of a test into the executable shorthand. It also produces tables from the SEARCH lists. The second stage selects the items that are used in a test from the global item bank (or banks) and creates a file containing only those selected items. The third stage makes a final pass on the executable shorthand code to resolve branching addresses (i.e., to fill in addresses at the branch-initiation points that were not known when that part of the source was consolidated on the first pass). It also replaces item references with file addresses (from the file created in stage two). The design for the test consolidator is depicted graphically in the hierarchy chart of Figure 6.

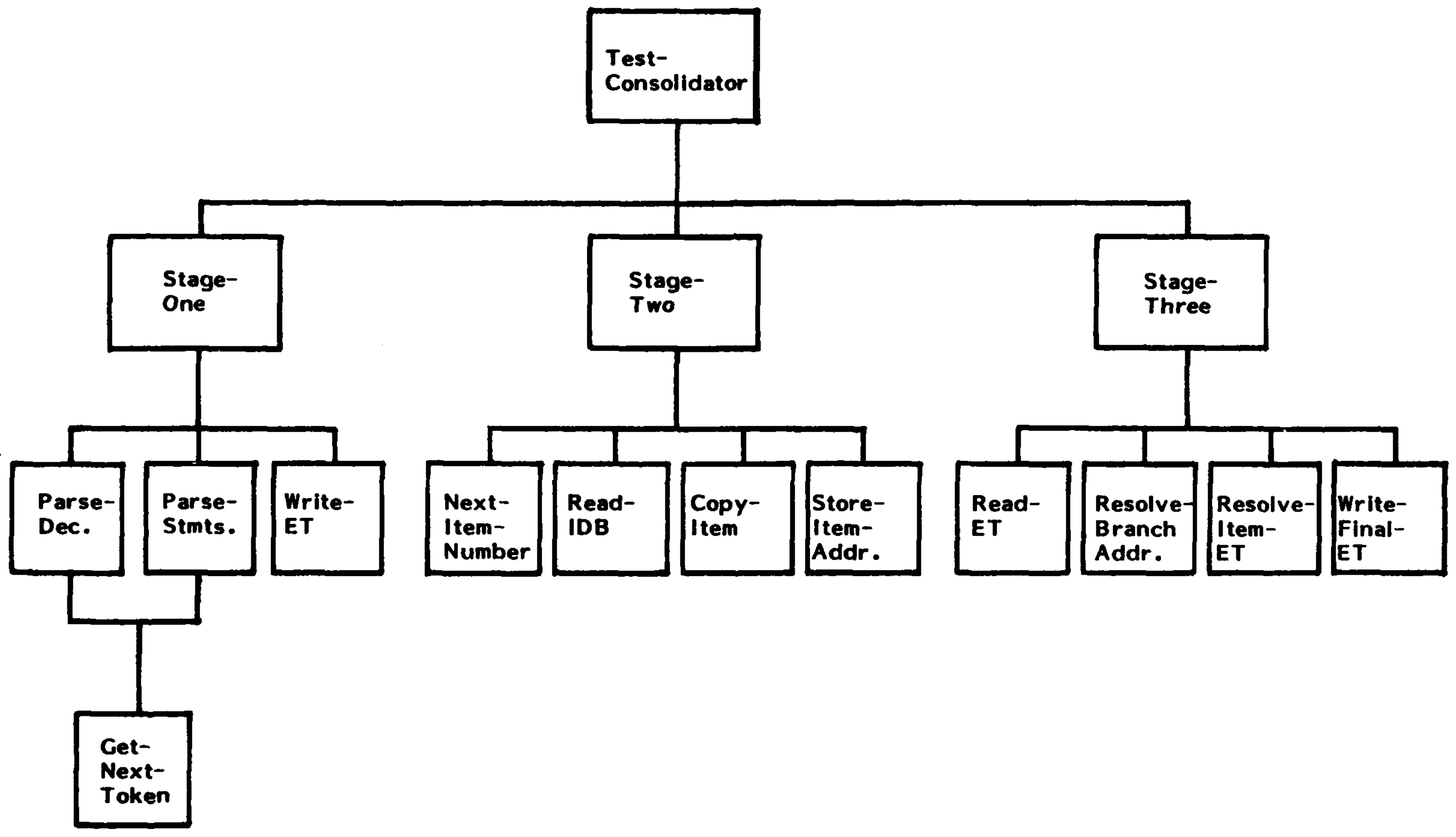
Stage-One. Stage-One is a parser for CATL statements. To parse a statement, the source statement from the test-specification file is translated into the shorthand executable form. When the statement has been parsed, the executable statement is written to the executable test file. This parsing is accomplished through a combination or two well-established parsing techniques. Most of the statements are parsed via a recursive-descent procedure. The parser design follows very closely the BNF (Backus-Naur Form, a system for formalizing syntax) definition of the language. (See Appendix B for a partial BNF description of CATL.)

For example, a CATL test consists of the following sequence: (1) the keyword TEST, (2) declarative statements, (3) executable statements, and (4) the keyword ENDTEST. Since the outermost syntactic unit is a test, the Stage-One parser attempts to find the four syntactic parts that make up a test. First it must find the keyword TEST. Finding that, Stage-One simply writes the appropriate keyword to the executable test file.

Next must come the declarative statements (see section 8.5 of the draft User's Manual in Appendix A). Since declarative statements can themselves consist of many parts, it is appropriate that Stage-One call a separate module to parse declarative statements. Hence, Parse-Declaratives is required.

Once declarative statements have been parsed, executable statements follow. These are defined in the BNF description as the construct

Figure 6. Consolidator Overview



<statements>. Again this is a complex structure, so a separate module, Parse-Statements, is called to parse <statements>.

The final element of a test is the keyword ENDTEST. When this word is encountered, the corresponding ENDTEST is written to the executable file.

The process described above follows directly from the BNF and exemplifies the design for Stage-One and its subordinate routines. The Parse-Declaratives and Parse-Statements modules can be designed in a like manner: Keywords are handled within the module for the corresponding syntactic element; more complete structures are handled by subordinate modules.

The second form of parsing used within the first stage of the test consolidator is operator-precedence parsing. This approach is used to parse logical and arithmetic expressions. Operator precedence is used here because is it especially well-suited to parsing arithmetic and logical expressions -- it can use the precedence and associativity of the various operators to guide the parse (Aho & Ullman, 1977).

The result of parsing an expression is a Reverse Polish Notation (RPN, or postfix) form of the expression. RPN is used as the executable form for expressions because it is fast and convenient to execute. It allows the expression to be evaluated during test administration by means of a simple stack mechanism.

Get-Next-Token is a utility routine called throughout the parsing process. It reads the source file (opening it if necessary) character by character until it has enough characters to comprise a token, the smallest syntactical element. Tokens are keywords (such as TEST), operators (such as = or +), and user-defined identifiers (such as item numbers or labels). Once a token has been found, Get-Next-Token looks it up in the appropriate symbol table to determine if it is a keyword, operator, or user-defined name. If it is a user-defined name, that name is installed in the symbol table for future reference.

Get-Next-Token also passes comments and processes continuation lines. Thus, the parsing routine sees the source file not as a file with lines of characters, some to be ignored (comments) and some to continue across lines, but rather as an input file consisting of a continuous stream of tokens.

The procedure Write-ET writes the ENDTEST keyword to the executable test file (ET). Additionally, it resets the consolidator to the condition it was in (e.g., to the previous test level, if the ended test was a subtest) prior to beginning the test.

Stage-Two. Stage-Two collects the items specified in the test and copies them from the item data base onto a separate file to be used during test administration. This approach makes the executable test a wholly separate entity, removing the restriction that the entire item bank be on-line during test administration.

Stage-Two begins by getting an item name from one of the tables built during the first stage. The Next-Item-Number procedure extracts this name from the table. Subsequent calls to Next-Item-Number will return subsequent names from the table.

The Read-IDB procedure locates the specified item in the item data base (IDB). It returns a file pointer/disc address for the location of that item in the IDB. Copy-Item takes that address and copies the item into the file of items for this test. It creates that file, if necessary (as on first call) and returns the address of the item in the new item file.

Store-Item-Address stores the address returned by Copy-Item in the table read by Next-Item-Number. This information will be used by the third stage to replace item numbers with disc addresses in the ET.

This process is repeated for all items in the table of item numbers. When all items have been copied, Stage-Two is complete.

Stage-Three. Stage-Three then makes a final pass of the ET. Read ET reads one symbol from the ET. If it does not need alteration, it is written immediately to the executable test file in its final form by Write-Final-ET.

There are two constructs modified by Stage-Three. First, any branch, whether implied (as with IF) or explicit, is given a file address to replace the branching location; this is accomplished using the procedure Resolve-Branch-Address. This address is the file address of the referenced instruction in the executable test file. The addresses were saved during Stage-One when the locations were encountered in the first pass through of the test. They were stored in the symbol table for labels (implied and explicit) as maintained by Get-Next-Token.

The other construct that Stage-Three deals with is item references. The file address for an item, saved by Store-Item-Address, is used here by module Resolve-Item-Address to replace the item number in the ET. This eliminates the need for directory searches during test administration. An item is located directly with this disc address.

Summary

CATL programs may be created by using the template processor or by using the system editor. After a CATL program has been created by either method, it can be consolidated using the test consolidator. The consolidator combines the testing strategies from the CATL program with the items from one or more item files, and produces an executable test file. The executable test file can be used with the test administration and test analysis subsystems.

Item Banking

The Item Banking subsystem is used to maintain files of items. The responsibilities of the item banking system are to provide a means of entering items into a random-access item bank, editing those items, and managing a table that allows the items to be accessed by item number.

The item entry and editing features of the bank will be accomplished by a special-purpose item editor. In design, this editor is simply a text editor. Its specific design should match as closely as possible the system editor on the chosen system. To do this, the system on which it is to be implemented must first be chosen. Since it is such a standard design and since its specific features are unknown, no specific design for the item editor is provided in this preliminary design document.

The random access portion of the banking system is a standard implementation of a keyed random access method (KRAM) filing system and requires no further explanation here.

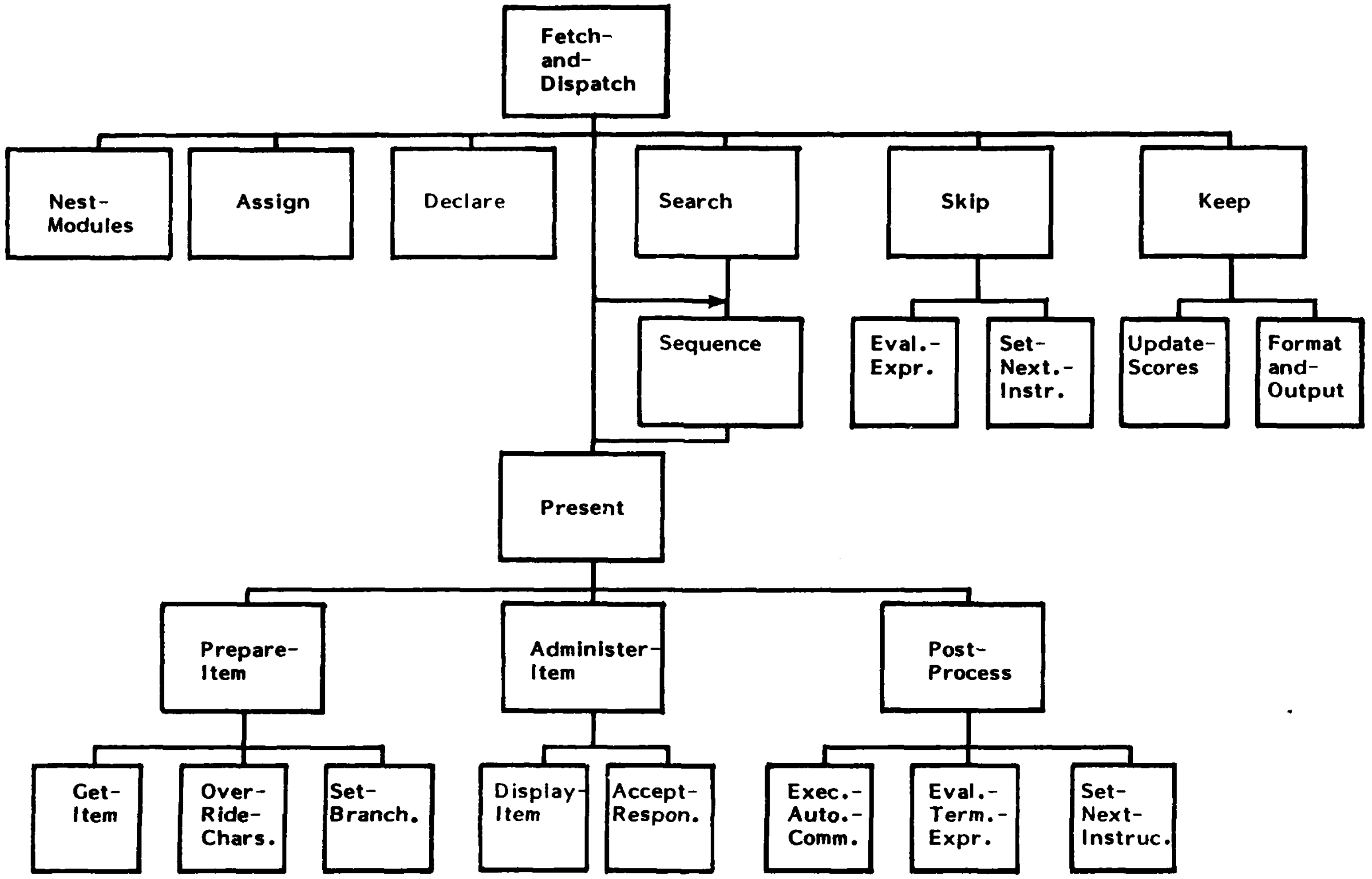
The item banking system designed for the proposed system should in no way be confused with the extensive item banking systems developed for computer-assisted test construction systems. These systems contain additional features for selecting, organizing, and formatting items into tests. Such features are unneccessary in a computerized testing system.

Test Administration

The Test Administration subsystem consists of the programs used to administer a test to an examinee. At the heart of the test administration software is the test executor, which interprets the test code, initiates item displays, accepts responses, and invokes scoring procedures. That is, it administers the tests created by the test consolidator, and through interaction with the examinee, creates a file of test results. These results can be used with the test analysis or test interpretation subsystems.

The executor software administers the test by executing the executable test file produced by the test consolidator. Figure 7 shows these functions as logical modules arranged in a hierarchy, with the highest functions on top and subordinate functions below. As is the case with all designs in this report, Figure 7 represents only the upper level of the design.

Figure 7. Executor Overview



Fetch-and-Dispatch

Fetch-and-Dispatch is the highest procedure in the hierarchy. It reads the keyword of a statement from the executable test file, determines which subordinate procedure is responsible for the instruction, and calls that procedure. After the statement has been executed, control is returned to Fetch-and-Dispatch to get another instruction.

Nest-Modules

Each test must begin with a TEST instruction, which can in turn signal the beginning of a subtest nested within the main test. When a TEST instruction is encountered within a test, the status of the test currently executing must be stored, and the set of variables appropriate for the new test must be activated. When an ENDTEST instruction is encountered, the opposite sequence must be performed: The variables of the current test must be inactivated and the status of the previous test must be restored. This is referred to as module nesting and is accomplished by the procedure Nest-Modules.

Assign

Assignment is accomplished by the Assign procedure, which is invoked with the SET statement. A SET statement consists of an expression and a variable name. The expression is evaluated and the variable is set to the result by Assign.

Declare

The Declare procedure is responsible for setting up status tables required by AUTOKEEP, SETSCORE, and TERMINATE. In response to an AUTOKEEP statement, it sets up a string of variables to be kept after each item is administered. In response to a SETSCORE statement, it sets up a correspondence table to assign variables to scoring algorithms. In response to a TERMINATE statement, it sets up a logical expression in a buffer to be used in deciding when to terminate the test. All information from the declarative statements is kept in a set of status tables. No overt actions are performed by the declarative statements. They affect test execution only through the action of other instructions that reference the status tables.

Present

Present oversees the actual presentation of the item. It can be invoked by an item statement or by several other statements that direct the presentation of an item. The item specified on a call to Present is read and prepared by Prepare-Item using Get-Item. Get-Item also gets any required pre- or post-messages. If branching is specified for the item, it is set up by Set-Branches. If item characteristics are specified, Override-Characteristics replaces the original values stored with the item with the new values.

After the item has been prepared, Administer-Item administers it. The item is sent to the examinee's terminal by Display-Item and the examinee's response is read and edited by Accept-Response.

After the item has been administered, Post-Process takes control. If the automatic command AUTOKEEP has been specified, the required scores are updated and output by Execute-Automatic-Commands. The logical expression stored in the termination variable is evaluated by Evaluate-Termination-Expression. Set-Next-Instruction then evaluates the termination result and the branch information to determine where execution should proceed.

Sequence

The procedure Sequence manages a list of item statements. It keeps a pointer and, each time it is executed, it causes the presentation of the next item in the list. It may contain a branch clause to cause branching after the item and it may contain characteristics to override characteristics previously overridden by those in the item statements.

Search

Search finds the item that will yield the most psychometric information given the current score status. To do this on an exhaustive, sequential basis would require too much computer time. Instead, the consolidator organizes the items in the search pool by trait level. Each trait level corresponds to a sequence of items. The executor's job is thereby reduced to determining a trait-level category and calling the proper sequence. The sequence used here is processed by calling the procedure Sequence.

Skip

Conditional execution of sections of CATL statements is implemented by the Skip procedure. Skip controls the flow of execution through the IF, ELSEIF, and ENDIF statements. The conditional instructions are followed by a boolean expression to be evaluated as true or false. If the expression is true, execution continues on the next line; otherwise execution is directed to another location. Evaluate-Expression evaluates the expression, and Set-Next-Instruction sets the location at which execution will continue.

Keep

The Keep procedure writes variables to a file for later use. The KEEP statement includes a variable list to be preserved. When executed, the Keep procedure writes this variable list to the file. To do this, it calls Update-Scores to update any score it will use and calls Format-And-Output to write the data to a file.

Test Interpretation

The Test Interpretation subsystem is used to present useful information based on test results. Using the system editor, the test developer creates an interpretation specification that defines the meaning of the results. The test interpreter uses the interpreter-specification and the test-result files (Figure 4) to generate a report on the interpretation of the test results.

The interpretive language is described in Chapter 9 of the draft User's Manual in Appendix A. It is based on the concept of an interpretive module consisting of a group of statements preceded by a logical expression. During execution of the interpretation, the expression is evaluated and if true, the associated module is executed. If the expression is false, execution continues with the next module.

As a test specification had two forms, source and executable, a test interpretation has two forms, source and executable. Thus, the interpretation system has a program to convert the source to an executable form and another program to execute the executable form. The former is called the Interpretation Compiler and the latter is called the Interpretation Executor.

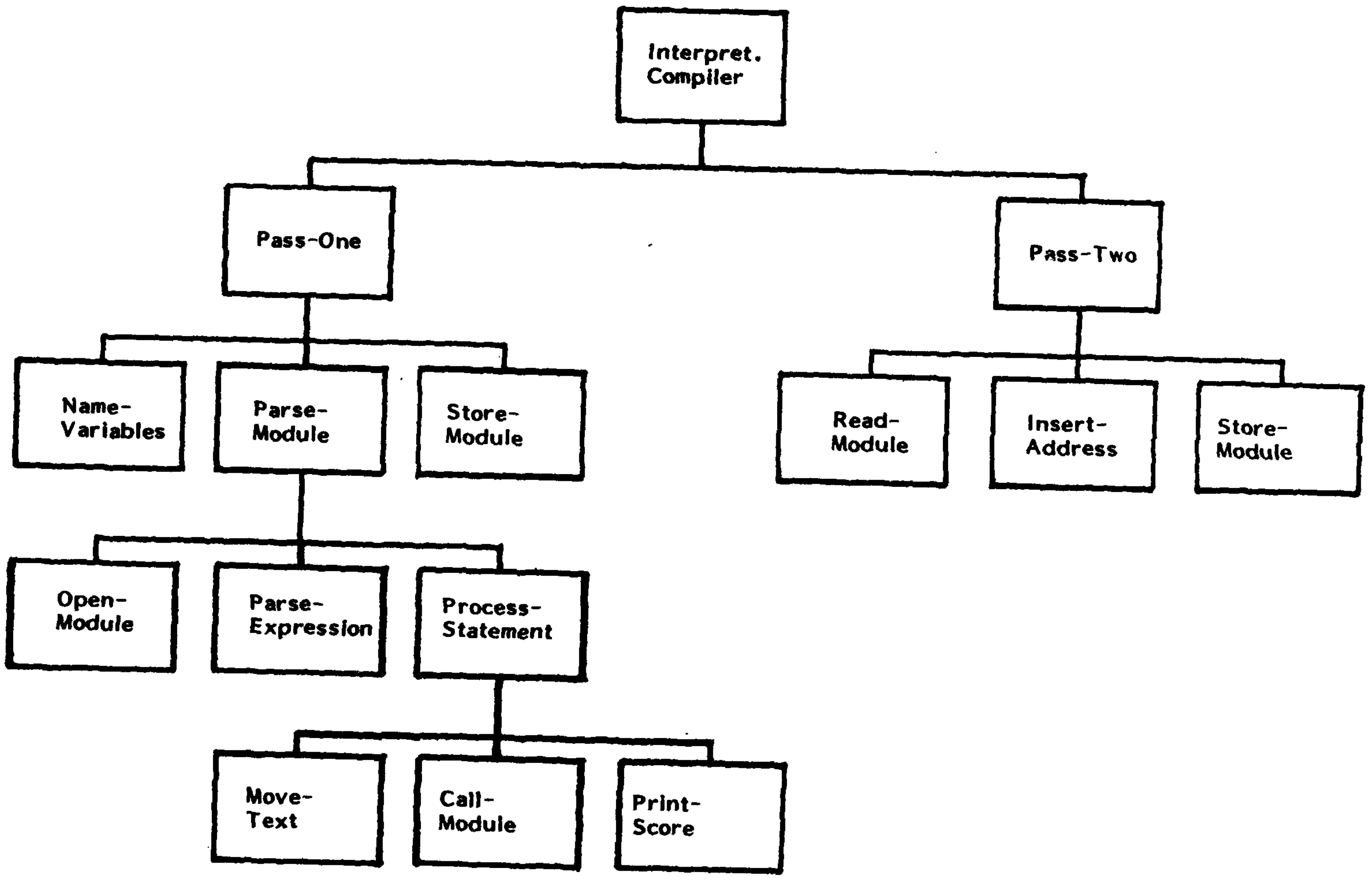
Interpretation Compiler

The Interpretation Compiler is a two-pass compiler that translates the source interpretation language into a convenient shorthand form. The compiler is diagrammed in Figure 8. Pass-One does most of the work of translation. Pass-Two inserts module addresses that are unresolved after Pass-One is completed.

Pass-One. Three procedures are at the first level of the Pass-One hierarchy. The interpretation system reads score data from files produced by the test executor. To refer to these scores symbolically, names must be associated with the values read. The procedure Name-Variables performs this task. Using a list contained at the beginning of the interpretation specification, Name-Variables associates symbolic names with data locations in the input file. These symbolic names are stored in a table along with a pointer to the score to which they refer so that the values can be retrieved during execution to evaluate the expressions.

Parse-Module is the procedure responsible for analyzing the module of interpretive text into its component parts for storage. It accomplishes this through three subordinate modules: Open-Module, Parse-Expression, and Process-Statement.

Figure 8. Interpretation Compiler Overview



Open-Module is called when a left bracket is found, indicating the beginning of the module. It reads the module name, if one is provided, and puts it into a table along with the module's address in the executable interpretation file so that it can be called by name by other modules.

Parse-Expression converts the logical expression at the beginning of the module to RPN form and stores it in an expression buffer. If no expression is provided with the module (an acceptable condition), a logical expression equivalent to "true" is inserted in the buffer so the module will be executed every time it is encountered.

Process-Statement processes the textual portion of the module (i.e., everything following the expression). It requires three subordinate modules to accomplish this. Process-Statement processes a single character at a time. If the character indicates no special function, Process-Statement calls Move-Text to move a character of text into the temporary module storage buffer. If the character indicates a call to another module, Call-Module is called to insert the keyword and either insert the address of the called module or reserve space and flag the location for resolution by Pass-Two. If a character calling for a score to be printed is encountered, Print-Score is called. Print-Score inserts the proper keyword and the table location of the desired score.

Upon completion of Parse-Module, Store-Module is called. Store-Module gets the address of the next free location in the executable interpretation file and writes the buffered module. Upon completion of the write, it updates the address of the next free location in the file.

When an end-of-file is encountered on the source interpretation file, Pass-One terminates and Pass-Two begins.

Pass-Two. Pass-Two has only to insert the addresses of the modules called by other modules in the specification that were not available when they were needed by Pass-One. It does this using three subordinate procedures.

Read-Module reads a module from the executable file and puts it in the module buffer. It essentially does the opposite of what Store-Module does in Pass-One.

Insert-Address scans the buffered module looking for flags indicating unresolved addresses. When it finds one, it looks up the indicated module in the table set up by Pass-One. If the module is referenced in the table, the address is inserted. Otherwise, an error message is issued.

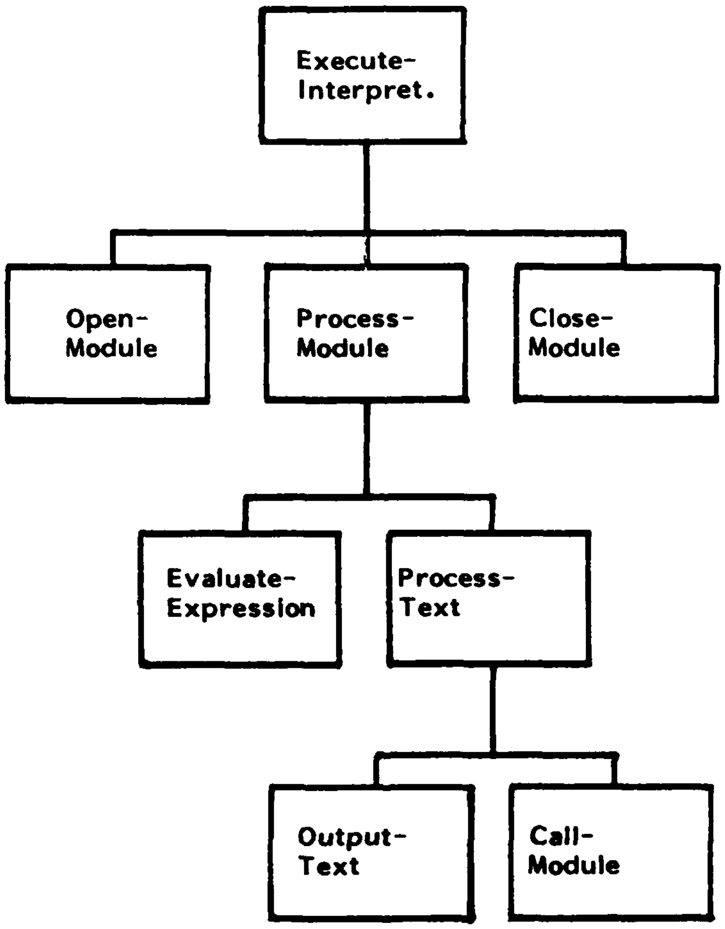
Store-Module is the same procedure used by Pass-One and performs the same function.

When the end of the executable file is reached, the compiler terminates. If any compilation errors were encountered, the executable file is flagged to prevent it from being executed. Otherwise, the file is ready to be executed.

Interpretation Executor

The Interpretation Executor, diagrammed in Figure 9, is somewhat simpler than the compiler. The main procedure, Execute-Interpretation, has three subordinate procedures: Open-Module, Process-Module, and Close-Module. The internal logic of Execute-Interpretation causes it to execute these three procedures sequentially. Since the modules can call each other, however, Execute-Interpretation may be called recursively from the procedure Process-Module.

Figure 9. Interpretation Executor Overview



Open-Module. To allow the remainder of a module to be skipped if the associated expression is evaluated to be false, the location of the end of the module must be known when execution of the module begins. This information is provided in the executable interpretation file by the compiler. The nature of the interpretation language provided allows nested modules, however. When a left bracket is encountered indicating the beginning of a module, the end location of the module currently executing must be stacked so that the module can be returned to. The function of Open-Module is to stack this information from the current module, if one is executing, and to initialize the next one. Complete stacking of the module is not required because the information prior to the module call has already been processed and will not be needed again.

Process-Module. Process-Module sequentially processes the interpretive information in the module. It uses two subordinate procedures to accomplish this. Evaluate-Expression evaluates the module's logical expression. If it is true, Process-Text is called. If it is false, control returns to Execute-Interpretation to close the module.

Process text has two subordinate procedures: Output-Text and Call-Module. Output-Text processes the text character by character. Standard characters are printed. Output control characters are translated into their appropriate functions. Score-printing characters cause a score to be printed. Call-Module generates a recursive call to the main procedure, Execute-Interpretation. Stacking of the required information is handled in that procedure.

Close-Module. Close-Module performs the reverse function of Open-Module. It unstacks the information from the previously executing module and continues where that one left off.

Summary. The interpretation system designed for the proposed system consists of a compiler and an executor. A very simple module-processing language is used to specify the interpretive output that is produced from a test record. This language is compiled to executable form and executed by the interpretation executor. The output produced may contain textual and numerical information about an examinee's test scores.

Test Analysis

The test-analysis subsystem provides analyses of the quality of the tests and the items. It consists of several related programs providing test evaluation, item pool refinement, and item calibration. These programs require information from the items, the test file, and the test results file.

The test-analysis programs, except for some file updating features, are very similar to others already discussed in relation to other parts of the system. They a e basically statistical programs and no system details for these programs are provided here.

Summary

The software of the proposed system is designed to perform most of the tasks that will be required by a researcher or practictioner wishing to do adaptive testing. Overall, the system is quite large, a neccessity if it is to be comprehensive. However, it is very modular in design. The modularity of the programs allows one to set up an adaptive testing system using only the programs needed.

The system is designed to address a wide range of expertise in both computers and adaptive testing. The system can be used simply to administer pre-developed tests. At the other extreme, it can produce a complex test for the user who understands the field and wishes to be creative. Provisions have been made for several levels of expertise between these two extremes.

IV. SELECTION OF COMPUTER HARDWARE

Hardware is the physical part of a computer. It includes a central processor, memory, CRT screens, keyboards, printers, and disc and tape drives. This section describes the requirements, evaluation, and selection of hardware for an adaptive testing system. No understanding of computer hardware is necessary to comprehend the hardware recommendations made in this chapter. However, knowledge of the following terms will allow the reader to understand the details of the evaluation which led to these recommendations.

The fundamental unit of computer memory storage is called a byte. A byte is the amount of storage needed to store one character of text or a number less than 256. The most common unit of storage is a K. A K is 1024 bytes. A megabyte (abbreviated M) is 1024K or 1,048,576 bytes.

Computers have two kinds of storage: random acccess memory (called RAM, or sometimes just "memory"), and secondary storage (usually floppy discs or hard discs). All programs run by a computer must be in RAM, but RAM is very expensive. Everything stored in RAM is destroyed when the computer is turned off. Disc storage (secondary storage) is much less expensive than RAM and is not affected by turning off the computer. Therefore, programs and other data are stored on disc when they are not needed in RAM. The amount of RAM in a system determines how large a program and how much data can be used at one time. The amount of disc space determines how many programs and how much data can be stored in total.

Microprocessors are very small computers. They are integrated circuits (sometimes called chips) approximately the size of a stick of gum. Two types of microprocessors are discussed in this chapter: eight-bit processors and sixteen-bit processors. Eight-bit processors are older, less expensive, slower, and can access a maximum of 64K of RAM. Sixteen-bit processors are newer, more expensive, faster, and can access more than 1M of RAM.

A central processing unit (CPU) is the part of a computer system that contains the microprocessor and often some or all of the RAM. A bus connects the CPU to the rest of the computer system. Many computers use the same kind of buses, so that components of one computer can be connected to other computers with the same type of bus. A computer that uses a popular bus is more flexible, because more compatible parts are available for it. The S-100 bus is the most popular bus for small computers.

A computer network is a group of computers that exchange information through communication cables.

Images on computer screens are composed of many small dots called pixels (short for picture elements). Color images are composed of combinations of pixels in the three primary colors. On black-and-white computer screens, shaded images may be displayed by using pixels of different intensities; this is called a gray scale.

Hardware Requirements

Hardware requirements were drawn from two sources. The first was the review of existing item types and testing strategies. The second was the survey of potential system uses. Both sources of information were used to determine the minimum hardware requirements and the optional hardware configurations desired for an adaptive testing system.

Cost

The minimum hardware configuration was designed to cost less than $3,000. List prices were used to compare the costs of different types and brands of hardware. Because most hardware manufacturers offer quantity discounts or governmental and institutional discounts, and because the cost of computer hardware is decreasing rapidly, a twenty-percent factor was added to the $3,000. In the evaluation of hardware, it was assumed that any computer hardware available at the time of the evaluation with a single-unit list price lower than $3,600 would be available for $3,000 by the time a prototype system was developed, especially if discounts were applied.

Display

Several survey questions were designed to determine the type of display required for adaptive testing systems. Eighty percent of those responding indicated that they needed a system to display black-and-white line drawings in addition to text. Less than one third were interested in color displays, moving displays, or a system that could only display text. Forty percent were interested in shaded drawings, and 53% were interested in a videotape or videodisc interface. Of the 30 testing strategies reviewed, 24 required a character display and 10 required a graphic display. A videotape or videodisc interface would be required for the three strategies that require dynamic display. To meet these needs, the display on the minimum configuration should include a text and graphics capability with a resolution from 256 x 256 to 512 x 512 pixels (dots). Options should include gray-scale display for shading and a video interface.

Input and Output Devices

Several survey questions dealt with the input devices needed for the system. Seventy-one percent of the researchers wanted a simplified keyboard, 32% wanted a joystick or trackball, and 50% wanted a touch screen. There was very little interest in voice recognition as an input device. Most of the researchers indicated that the system would receive unusual wear and tear or abuse by its users. Sixty-seven percent thought that the buttons would be pressed harder than necessary. Since users have more contact with the input devices than with any other part of the system, the input devices should be especially durable. Twenty-eight of the testing strategies reviewed require a keyboard, although a touch screen could be substituted for a keyboard in most strategies. Two strategies require a joystick. Voice recognition was considered desirable, but not necessary, for any strategies. A standard keyboard would be required, at least for test development. Special function keys (eg, a numeric keypad) may be able to simulate or replace a simplified keyboard, but a joystick may still be required on many systems for analog input. The minimal configuration should include a durable keyboard with special function keys. Options should include joystick, touchscreen, and simplified keyboard.

Less than one third of the researchers surveyed were interested in voice output or variable-frequency tones. These features should probably be available as options.

Network Capability

In the survey, several methods of administering tests were suggested. Seventy-seven percent of the researchers indicated a need for a proctor's station in addition to the testing stations. This would require either a central processor to which all the stations would be connected, or a network to allow the proctor's station to communicate with the testing stations.

Compatibility with Other Software

Of the researchers surveyed, 63% were interested in also using this system to run compilers, accounting systems, or word-processing systems. Although this is primarily a software concern, the hardware must support these general-purpose software packages. Since most of the available software packages run under one of the popular operating systems (such as CP/M or UNIX), this hardware should include timers, interrupts, and other features required to run at least one of these operating systems.

Secondary-Storage Requirements

Secondary-storage requirements must be taken into consideration when selecting a computer system. However, very few people, testing researchers included, know how much disc space is needed for various items. Therefore, the researchers were asked to estimate the size of their largest item pool and the number of words in the text of these items. The required amount of secondary storage was estimated by the following equation:

STORAGE = (max. items x (words per item x 6 + parameters))
+ fixed storage

Averaging the responses and using probable figures for parameters and fixed storage, the average maximum storage requirement was estimated to be near one megabyte of disc storage. This is near the maximum capacity of two high-density floppy discs. If floppy discs are used, some limits will have to be placed on the number of items in large item pools. The smallest hard discs hold five times as much space, and could easily contain the largest item pools. Hard discs would decrease the waiting time between items, but should remain optional because of their cost.

Performance Requirements

Most of the modern testing models require a fair amount of computing power, yet most of the researchers surveyed indicated that an adaptive testing system should have a delay of no more than a few seconds between items. Under these conditions a numerical processor may be required to speed up the testing model calculations.

Summary

In conclusion, the following features are required for the minimum configuration of an adaptive system:

1. Character display
2. Black-and-white graphics
3. Durable keyboard
4. Simplified keyboard or function keys
5. Ability to run compilers and other general software
6. Ability to connect to a proctor's station
7. Disk storage approaching one megabyte

The following features would be desirable options:

1. Gray scale display for shaded displays
2. Videotape or disc interface
3. Joystick
4. Touchscreen
5. Hard disc
6. Numerical processor

System Development Approaches

Several approaches could be taken to develop a computer system that would meet the requirements for the adaptive testing system defined above. Mainframe computers and minicomputers are too expensive to be used in an adaptive testing system costing less than $3,000. It would be possible to use a mainframe computer or minicomputer in a multi-user system costing less than $3,000 per user, although this implementation would make systems for a small number of users more expensive. Large systems would also be very expensive for test development where only one console is required. A microprocessor-based system capable of supporting a single user can cost less than $3,000. Microprocessor-based systems are portable, and they do not require the air conditioning and special facilities that larger computers require. Furthermore, the addition of testing stations does not degrade the performance of the system, nor does it affect the other stations if one console breaks down. A review of the state-of-the-art in computer hardware by Croll (1980) showed that both microprocessors and minicomputers were capable of supporting CAT interactive testing and monitoring functions.

Many types of microprocessors available today are based on the most advanced computer technology. They are fairly powerful and very inexpensive. Microprocessors are designed for a broad range of applications, including input/output functions and general computations. Although it would be possible to design and build a special microprocessor for adaptive machines, this expensive task is unnecessary since most of the 8- and 16-bit microprocessors available are well suited to adaptive testing applications.

Given that microprocessors will be used, it remains to be decided how the computer system built around a microprocessor will be obtained. The following paragraphs discuss the advantages and disadvantages of several options, including building a system from components and purchasing a complete system.

Build from Chip-Level Components

A system could be built that would contain exactly those components needed and nothing more. This would minimize the cost of the system by eliminating the unneeded components that would exist in a purchased system. This option would also allow the inclusion of components unique to adaptive testing systems that would not be available on general-purpose computers.

However, building a system from the component level would be very expensive. Power supplies, keyboards, disc drives and hundreds of other parts would have to be sampled, evaluated, and selected. Circuit boards would have to be designed, tested, and modified. A case would have to be manufactured. Parts would have to be stocked and second sources established. Because of the low volume, it would be necessary to pay high prices for parts, especially custom-made parts. This project would be equal to the development of a personal computer in every way. However, the high cost of development could not be spread out over a large number of units sold. Like all new products, initial releases would have a few undetected problems that would be encountered by the first users.

Microprocessor technology is changing rapidly. Microprocessors in use today will be replaced by faster and more powerful microprocessors in only a few years. As a result, any microprocessor-based system built today will soon be inferior to new systems unless it is redesigned every few years; this is what personal and small business-computer manufacturers must do to remain competitive. This constant redesign is extremely expensive; however, the alternative is to produce a system that will soon be obsolete. Steffen, Gray, Wasmund, and Lanos (1978) describe an effort to develop a much less ambitious system from chip level components. Their system, targeted for $500 four years ago, is probably no longer cost effective for the capability required.

Thus, the option to build from components has very high overhead and a long lead time, and results in a product that is functionally equivalent to existing products. In addition, the product would have a lifetime of only a few years before it became obsolete, unless funds were allocated to constantly redesign it.

Start with a Single-Board Computer

Single-board computers contain all the electronic components necessary for a small microprocessor-based system. They can be purchased alone or in a case with disc drives, power supply, and a terminal (or with an attached keyboard and screen). Single-board computer systems are relatively inexpensive because all the electronic components are packed onto one board. Many brands have been thoroughly tested and are considered highly reliable. These boards are available immediately, and redesigned boards are usually available soon after new microprocessors become popular.

Because all the components are on one board, many single-board computers cannot be expanded, however. It is often impossible to add more memory, even though it may be required by the software. Single-board computers often have only one basic configuration, so it is impossible to offer options like graphics.

The single-board computers that are sold in a case with disc drives usually do not contain any kind of bus. If a system has a bus that is accessible through a non-standard connector, then only the company manufacturing that board can provide expansions. Some companies offer many options to expand their system, but many companies do not.

Use a Packaged System

The needs of an adaptive testing system are almost identical to the needs of any personal or small business computer. Both systems must calculate equations at high speeds, read and write information to disc files, display information on the screen, and accept responses from the operator. The electrical and packaging requirements are also identical. The special requirements for adaptive testing (high-speed calculations, data base processing, and graphics) are the same as those available on general-purpose microprocessor systems. There are very few items that could be eliminated for an adaptive testing system. In addition, many personal-computer manufacturers have established service centers in major cities across the country, so repairs can be made quickly.

Personal computers are well tested, and because they are manufactured in very high volume, they are the lowest-priced computers available. Since they are designed to be flexible, many configurations that will meet the needs of different adaptive testing applications are possible.

A major drawback involved in using a packaged system is that personal computers are designed primarily for stand-alone operation. The survey showed that many potential users want a test-monitoring station that can display information on all of the testing stations. Some external device must be used to connect the personal computers into a network.

Use a Packaged System with a Network

A personal or small business computer meets most of the needs of an adaptive testing system. The only major deficiency is that it lacks a way to communicate with a proctor's console. A minor drawback may be inadequate disc space. Responses to the survey indicated that about one million bytes of disc space may be needed for the largest item pools. Although discs of almost any size can be connected to a personal or small business computer, $3,000 computers rarely have more than one million bytes.

To solve both of these problems, a local network could be used to connect the stations. The proctor's station could send information to the testing station to signal it to start a test. The testing station could send back information about the status of the examinee. Each station would still be self-contained and could be used without the network. If the network were to fail, proctors could re-start tests by using special control commands on each testing station.

Many of the networks available contain a hard disc that can be acccessed by any of the stations connected to the network. Even one of the smallest hard discs should be able to hold several of the largest item pools with space left for test storage.

Current Computer Systems

To select a packaged microprocessor-based computer system that met the requirements of adaptive testing, a list of available systems had to be compiled. To do this two sources of information were used.

Webster's Microcomputer Buyer's Guide (Webster, 1981) has a very good list of systems and very detailed information about their options and costs. However, there were two problems with the information in this guide. First, because it was published in 1981, many of the newest computer systems were not listed. This is a serious shortcoming, because the newer systems are more powerful and less expensive than the older systems. Second, some of the information listed was already out of date; this was especially true of prices.

To obtain information about computer systems that were introduced in the last year, technical journals were researched. Issues of Byte, Electronics, EDN, Mini-Micro Systems, and Office Products News issued between August 1981 and May 1982 were scanned. Products listed in articles or advertisements that seemed to meet the adaptive-testing requirements were added to the list of available systems. In cases in which there was insufficient information to determine whether the system met the requirements, the system was added to the list anyway.

To verify the information in the microcomputer guide and to obtain additional information about products listed in journals, requests for information were sent out to the manufacturers of the products included in the list of available systems. A list of all systems considered is provided in Appendix C.

Systems That Were Too Expensive

One of the most critical requirements for the hardware was that it cost less than $3,000. Because the price of computer systems is decreasing, and because rules of supply and demand often cause price fluctuations, systems within 20% of $3,000 were included for initial consideration. Although an attempt was made to seek information only on those systems that appeared to cost less than $3,600, many of the systems about which information was received were clearly too expensive. (For a complete list of the systems eliminated because of cost, see Appendix C.)

Most of the systems based on sixteen-bit microprocessors were too expensive. Sixteen-bit systems are faster and can access more memory than eight-bit processors, but they usually require more memory, and the integrated circuits themselves are more expensive. Except for the Callan Data System and the Texas Instruments DS990, all of the sixteen-bit systems that were eliminated were designed to be upgraded to a multi-user system. These systems might be useful in applications where testing stations could be grouped into sets of three or four dependent systems. Note that several of these systems have processors that make the system software compatible with either minicomputers or smaller microcomputers.

Several of the sixteen-bit systems that were excluded because of high cost have special features that might make them ideal for some applications. The Chromatics system has extremely advanced color graphics capabilities. It can display graphic images in 256 colors on a

1024 x 768 pixel (dot) screen. It includes graphics software and has both joystick and light pen options. This system could be used to administer tests with almost photographic illustrations in the items. Unfortunately, very few testing applications can justify the cost of $20,000 per station.

Another sixteen-bit system with special capability is the Action Computer Enterprises Discovery. This system has many of the advantages of both shared minicomputers and stand-alone microcomputers. Like many of the sixteen-bit systems, it was designed to be a multi-user system. Unlike most multi-user systems, this system provides each user with a separate microprocessor and CPU board. All the CPU boards share discs and other devices within one package. Since each user has a separate microprocessor, the users do not slow each other down. However, if one of the shared parts of the system becomes inoperative, none of the users is able to function.

Most of the eight-bit systems that were eliminated cost approximately $4,000. The differences between these and the ones that were not eliminated were usually minor. Some of them have fancier packages; some have extra ROM (read-only memory) or extra board slots. The Business Operating Systems computer has an arithmetic processor for high-speed calculations. Quasar has 2.4 million bytes of floppy disc space. Several systems can be configured as a multi-user system.

Several of the eight-bit systems that were eliminated because of high cost have features that would make them ideal for some applications. The Intelligent Systems 8052 and 8053 have very good color graphics compared to the other systems in this price range. The MicroDaSys Millie has high resolution black-and-white graphics (640 x 400 pixels). This system is ideally suited to all of the adaptive testing requirements except cost (the Millie costs $10,000). Ithaca Intersystems produces a good eight-bit system with a memory management unit. Memory management is used in most large computers to increase the speed of running multiple programs or for multiple users. This is rarely found in eight-bit systems. Micromation and OSM Computer's Zeus were designed for multi-user applications. Micromation uses a network to allow all its stand-alone systems to act as a single system. Zeus works like the Discovery system described earlier; each system is shared among the users. If a shared part of the system does not function, none of the users can function.

Systems That Did Not Meet Requirements

The adaptive testing software described in this report requires large amounts of RAM and disc space. Survey results also indicated a strong desire for a system that can run general-purpose software. (This requires a common general-purpose operating system.) Any system on the list of available computer systems that failed to meet at least two of these three requirements was eliminated from further consideration. A common operating system was considered to be one that ran on machines from at least two manufacturers. The minimum cut-off points used for storage were 48K for RAM and 300K for disc storage.

Many of the systems with more than one configuration had an unacceptable configuration for less than $3,600 and an acceptable configuration for more than $3,600. In these cases the acceptable configuration was considered and eventually eliminated because of cost.

Several systems were eliminated because they lacked serviceability or flexibility. Most of the systems eliminated in this category can be characterized by two properties. First, they are marketed directly by the manufacturer and do not have dealers or local service centers. If any piece of the system fails, it has to be shipped back to the factory for repair. One manufacturer recommended that four weeks be allowed for all repairs. Systems of this type might be acceptable for organizations that do their own repair work. Second, to minimize costs, most of these systems have all the electronics on one board, and they do not have a bus. A bus allows optional RAM, discs, or other devices to be added to the system. The lack of a bus decreases the flexibility of the system by limiting the number of available options.

However, many of these systems have a better cost-per-value ratio than most of the products on the list of available computer systems. This is because their specific inflexible design allows a high optimization by removing all unnecessary parts. Were it not for the great need for flexibility and serviceability, one of these systems might have been recommended for acceptance.

Systems That Were Acceptable

The systems in this category met all the minimum requirements for adaptive testing applications. All of these systems were compared to choose the ones with features that are most desirable for adaptive testing applications.

Of these acceptable systems, some were found to be less desirable than others. The Radio Shack Microcomputer Technology Mod III has only 16 lines of text display and lacks the sophisticated graphics capabilities of the other systems. The Xerox 820 was designed to be a word processor and therefore lacks the flexibility of some of the other systems considered in this category. The Apple II computer has only 48K of memory, a slower microprocessor, and limited disc space compared to the others. The Televideo Systems TS802 is adequate in many respects, but lacks the flexibility and graphics capabilities of the systems finally selected.

Three systems remained that met all or most of the criteria described as important by the survey respondents. These systems are described in detail below. (The order of the subsections does not indicate any ranking or preference.)

Intertec Data Systems SuperBrain. The Intertec SuperBrain, like many small computers made by large manufacturers, is available in a number of configurations. The configuration of greatest interest to this project is the SuperBrain QD, which, like most systems examined, contains 64K RAM. It also has 700K disc space on floppy disc, a distinct advantage. The SuperBrain uses two Z80 microprocessors that will also run programs written for the 8080 microprocessor. The SuperBrain comes with the CP/M-80 operating system and utilities. More general-purpose software is available for the CP/M operating system than for all other microprocessor operating systems combined.

The SuperBrain also has several hard disc options from which the user can choose, including the SuperFive and SuperTen, which have five and ten megabyte Winchester hard discs, respectively. Larger hard discs for the system are available from Corvus.

Because it has the S-100 bus adaptor, it can use any of hundreds of external devices. Since many of the users surveyed wanted various options, this could be a great advantage for expansion. The standard configuration has two serial ports, and any device requiring such an interface could be connected.

Although the original SuperBrain did not have high-resolution graphics, it did have special characters that could be used for character graphics. This may suffice for many applications requiring low resolution graphics in its standard 24-line by 80-column screen. The recently introduced SuperBrain II is said to include more impressive graphics capabilities (at a lower price).

The list price for the system is $3595. There is also a configuration with 350K of disc space for $2695. CMC International, a wholesaler for this system, lists government/institution prices of $2950 for the first configuration (700K disc space) and $2565 for the second (350K disc space).

Intertec also offers a configuration called the Compustar Multiuser Microcomputer System. As many as 255 terminals can be used as work/test stations connected to a major, central terminal. The test stations would each have an internal data base and would be operating somewhat independently of the single Winchester disc to which they would all be connected. This would preserve the integrity of each test station in case of failure, but would otherwise provide each with the resources of the common hard disc for tests or item pools.

The list price of the Compustar unit configuration is $3995. However, since most of the disc space held in common can be used by each test station, a more simple test station can be configured for $2495. CMC International offers governmental and institutional rates on the first Compustar system of $3395 and the second, simpler configuration for $2125. (All prices are quoted for a single system. Further discounts are available for sales in quantities of three or more systems.)

IBM Personal Computer. The IBM Personal Computer is based on the 8088 microprocessor, which makes it much faster than the Intertec and other eight-bit systems. It also permits configuration with four times as much memory, if needed. The standard configuration of the IBM Personal Computer has 64K RAM and 320K disc space. The screen holds 24 lines by 80 columns of text. It supports two types of graphics: four-color graphics (resolution of 320 x 200 pixels) and black-and-white graphics (resolution of 640 x 200 pixels). This provides the kind of line drawing capability in which potential users expressed interest.

Hard discs could be purchased through several companies; Corvus Systems sells them in 6MB, 11MB, and 20MB sizes. The IBM Personal Computer also has a special interface for game adaptations which may provide a low-cost joystick option.

Although IBM designed a unique bus, the company's products are so well known and widely used that dozens of other companies are now manufacturing devices compatible with this bus.

A built-in time-of-day clock run by battery allowed the computer to time any software function automatically. This could be a valuable feature for testing applications.

IBM is also planning to offer an 8087 arithmetic processor option that would increase the speed of floating-point operations by a factor of one hundred.

A single-drive 64K system can be purchased for $2935. An additional drive brings the price to $3585 and provides for 320K altogether.

NEC APC (Advanced Personal Computer). NEC manufactures two small computers, the best of which was considered for recommendation.

The NEC APC has most of the advantages of the IBM Personal Computer. RAM in the initial configuration is 128K and disc space exceeds one megabyte. It is based on the 8086 microprocessor, which is slightly faster than the 8088 IBM used. But the two are totally software-compatible, so that applications designed for the IBM can be adapted to the APC.

One of the unique features of the APC is that it has two software-loadable character sets. This means characters of any shape may be created and used within the resolution of the 8 x 19 pixel matrix. In high-resolution graphics, the system can display drawings with a resolution of 640 x 475 pixels in eight colors. The system also has 22 software-definable function keys. This can be extremely useful in the development of tests (pass, help, multiple choice, etc.).

As with the IBM computer, CP/M-86 and MSDOS operating systems are available, making many off-the-shelf software packages feasible.

This system offers much more RAM and disc space for a lower price than the IBM Personal Computer. No information has been released about its bus, however. The primary mechanism for adding options to this system is through its two serial ports. These can be used for joystick, digitizer, or for connecting test stations to a proctor station.

The standard system is available with a time-of-day clock, a variable frequency tone generator, and an optional 8087 arithmetic processor ($250).

The standard system with black and white display is $3298. With two megabytes of disc space, the price is $3998, and with color, $4998.

Conclusions

A number of acceptable systems were found. The best of these were outlined above. Unlike most technologies, microprocessor technology is moving at a pace that can be measured in days and weeks, rather than months or years. This means that comparable or superior small systems may become available as this report goes to print.

V. RECOMMENDATIONS FOR FURTHER DEVELOPMENT

Accomplishments in Phase I

Phase I of this project began with four objectives:

1. To evaluate the need for a microcomputer-based CAT system and to specify the system requirements to meet that need.

2. To develop a preliminary system configuration to meet that need.

3. To develop a preliminary design for the software required by this system.

4. To document the system for potential users.

The first objective was accomplished by surveying both published literature and potential users to determine what types of items and tests the system should accommodate and what special features it should contain. The results of this survey, were used to determine required storage space, processing speed, item display formats, and response modes. The disc storage requirement was determined to be just under one megabyte. Required processing speed was defined to be the speed necessary to administer a Bayesian test and maintain a system response time of less than one second. Graphic displays and videodisc interfaces were suggested to accommodate varying item formats. Several response modes in addition to a standard keyboard were recommended.

After considering of the alternatives, a decision was made to configure the testing system around an existing microcomputer system. All microcomputer systems with list prices less than $3,600 were evaluated using a set of hardware criteria. Several were found to be acceptable for a CAT system. Of these three were selected for further consideration.

To meet the third objective, a top-level design for the system software was developed. This design was divided into five subsystems. The design of two of these subsystems -- the Item Banking and Test Analysis -- was considered to be simple at the top level and was not developed further in this phase. The other three subsystems -- Test Construction, Test Administration, and Test Interpretation -- were detailed at the top level.

Finally, addressing the fourth objective, a draft of a user's manual for the proposed system was developed. This manual was distributed among a small group of potential users for review. Their reviews were quite positive and their suggestions for revision were noted for incorporation into the final design and manual.

Recommendations for Phase II

The basic design of a microcomputer-based CAT system has been completed. Response to the design from potential users has been very favorable. A market for such a system obviously exists: The 50 potential users who responded to the questionnaire provide a market for 1,000-4,000 systems. Phase II should be directed toward the development and implementation of a small number of systems to determine how well the operational needs of the users match with the needs they reported in the survey.

Phase II should proceed in several stages. Demonstration models of the three computer systems selected should be acquired. Acceptance testing that includes exposure to an actual testing environment should be initiated using a small-scale testing system (perhaps one that administers only one hard-coded test).

At the same time, system design should proceed at increasingly detailed levels until the complete design is developed from the top down.

When the design and acceptance testing are complete, the software should be developed and implemented on the most promising of the systems. Basic hardware options should be interfaced and the neccessary interface software should be developed.

A system evaluation plan calling for on-site implementation of a few systems should be planned. (The questionnaire responses suggested that several respondents would like to participate.) The evaluation plan should consider all potential problems likely to surface in a commercial implementation of the system, including system capability and reliability, security problems, and user difficulties.

A few systems (e.g., 10) should then be installed at selected sites for evaluation. After a trial period of perhaps six months, the adequacy of the system should be re-evaluated. Any necessary redesign should take place at that time.

Finally, after the evaluation and any neccessary redesign work are completed, a plan for marketing the systems should be developed. This plan should consider all aspects of marketing from final targetting of the user population to plans for continuous updating of the system to keep it technologically current.

At that point, a low-cost, highly flexible and powerful commercial system for adaptive testing will be available. The promise of more efficient measurement through CAT technology will be one large step closer to reality.

REFERENCES

Aho, A. V., & Ullman, J. D. Principles of Compiler Design. Reading: Addison-Wesley, 1977.

Angoff, W. H., & Huddleston, E. M. The multi-level experiment: A study of a two-level test system for the College Board Scholastic Aptitude Test (Statistical Report 58-21). Princeton, NJ: Educational Testing Service, 1958.

Barrett, G. V., Alexander, R. A., Doverspike, D., Cellar, D., & Thomas, J. C. The development and application of a computerized information-processing test battery. Applied Psychological Measurement, 1982, 6, 13-29.

Bayroff, A. G., Ross, R. M., & Fischl, M. A. Development of a programmed testing system (Technical Paper 259). Arlington, VA: U.S. Army Research Institute for the Behavioral and Social Sciences, Individual Training and Manpower Development Technical Area, December 1974.

Bejar, I. I. Videodiscs in education: Integrating the computer and communication technologies. Byte, June 1982, p. 78.

Bell & Howell. GENIS I User's Manual, Bell & Howell, 1979.

Birnbaum, A. Some latent trait models and their use in inferring an examinee's ability. In F. M. Lord & M. R. Novick, Statistical theories of mental test scores. Reading, MA: Addison-Wesley, 1968.

Bock, R. D. Estimating item parameters and latent ability when responses are scored in two or more nominal categories. Psychometrika, 1972, 37, 29-51.

Brown, J. M., & Weiss, D. J. An adaptive testing strategy for achievement test batteries (Research Report 77-6). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, October 1977.

Church, A. T., & Weiss, D. J. Interactive computer administration of a spatial reasoning test (Research Report 80-2). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, April 1980.

Cliff, N. Complete orders from incomplete data: Interactive ordering and tailored testing. Psychological Bulletin, 1975, 82, 289-302.

Control Data Corporation. Control Data PLATO Author Language Reference Manual. St. Paul: Author, 1978. (a)

Control Data Corporation. Control Data PLATO CMI Author's Guide. St. Paul: Author, 1978. (b)

Cory, C. An evaluation of computerized tests as predictors of job performance in three Navy ratings: I. Development of the instruments (ONR-335, Letter Report). San Diego, CA: Naval Personnel and Training Research Laboratory, March 1973.

Cory, C. Relative utility of computerized versus paper-and-pencil tests for predicting job performance. Applied Psychological Measurement, 1977, 1, 551-564.

Cory, C. Interactive testing using novel item formats. In D. J. Weiss (Ed.), Proceedings of the 1977 computerized adaptive testing conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, July 1978.

Cudeck, R. A., Cliff, N., & Kehoe, J. F. TAILOR: A FORTRAN procedure for interactive tailored testing. Educational and Psychological Measurement, 1977, 37, 767-769.

de Finneti, B. Methods for discriminating level of partial knowledge concerning a test item. The British Journal of Mathematical and Statistical Psychology, 1975, 18, 87-123.

DeWitt, L. J. Personal communication, 1976.

DeWitt, L. J., & Weiss, D. J. A computer software system for adaptive ability measurement (Research Report 74-1). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, January 1974.

Ferguson, R. L. The development, implementation, and evaluation of computer-assisted branched test for a program of individually prescribed instruction (Doctoral dissertation, University of Pittsburgh, 1970). Dissertion Abstracts International, 1970, 30, 3856A. (University Microfilms No. 70-4530)

Fletcher, J., & Ree, M. J. Armed Services Vocational Aptitude Battery (ASVAB) correlational analysis, ASVAB Form 2 versus ASVAB Form 5 (AFHRL-TR-76-70). Brooks Air Force Base, TX: Air Force Human Resources Laboratory, October 1976.

Hanna, G. S. Incremental reliability and validity of multiple-choice tests with an answer-until-correct procedure. Journal of Educational Measurement, 1975, 12, 175-178.

Hunter, D. R. Research on computer-based perceptual testing. In D. J. Weiss (Ed.), Proceedings of the 1977 computerized adaptive testing conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, July 1978.

Kalish, S. J. A model for computerized adaptive testing related to instructional situations. In D. J. Weiss (Ed.), Proceedings of the 1979 computerized adaptive testing conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, September 1980.

Kingsbury, G. G., & Weiss, D. J. An adaptive testing strategy for mastery decisions (Research Report 79-5). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, September 1979.

Knerr, B. W. Problems of performance measurement in computer-based simulation. In D. J. Weiss (Ed.), Proceedings of the 1977 computerized adaptive testing conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, July 1978.

Krathwohl, D. R., & Huyser, R. J. The sequential item test (SIT). American Psychologist, 1956, 2, 419.

Lord, F. M. Robbins-Monro procedures for tailored testing. Educational and Psychological Measurement, 1971, 31, 3-31. (a)

Lord, F. M. The self-scoring flexilevel test. Journal of Educational Measurement, 1971, 8, 147-151. (b)

Lord, F. M. Applications of item response theory to practical testing problems. Hillsdale, NJ: Erlbaum, 1980.

Luker, M. The Minnesota Instructional Language, MIL Reference Manual. Minneapolis: University of Minnesota, Consulting Group on Instructional Design, 19`9.

McCormick, D., & Cliff, N. TAILOR-APL: An interactive computer program for individual tailored testing. Educational and Psychological Measurement, 1977, 37, 771-774.

McKillip, R. H., & Urry, V. W. Computer-assisted testing: An orderly transition from theory to practice. In C. L. Clark (Ed.), Proceedings of the first conference on computerized adaptive testing (PS-75-6). Washington, DC: U.S. Government Printing Office, 1976. (Superintendent of Documents Stock NO. 006-00940-9)

Myers, G. J. Reliable software through composite design. New York: Petrocelli, 1975.

Owen, R. J. A Bayesian approach to tailored testing (Research Bulletin 69-92). Princeton, NJ: Educational Testing Service, 1969.

Owen, R. J. A Bayesian sequential procedure for quantal response in the context of adaptive mental testing. Journal of the American Statistical Association, 1975, 70, 351-356.

Prestwood, J. S. Design of an examinee monitoring station. In Proceedings of the 22nd Annual Conference of the Military Testing Association, Vol II. Toronto, Ontario, Canada: Canadian Forces Personnel Applied Research Unit, October 1980.

Reckase, M. D. An interactive computer program for tailored testing based on the one-parameter logistic model. Behavior Research Methods and Instrumentation, 1974, 6, 208-212.

Ree, M. J., Mathews, J. J., Mullins, C. J., & Massey, R. H. Calibration of Armed Services Vocational Aptitude Battery Forms 8, 9, and 10 (AFHRL-TR-81-49). Brooks Air Force Base, TX: Air Force Human Resources Laboratory, February 1982.

Robinson, M. A., & Walker, C. L. The decision measurement system as a means of testing performance by simulation. In D. J. Weiss (Ed.), Proceedings of the 1977 computerized adaptive testing conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, July 1978.

Samejima, F. Estimation of latent ab[illegible]y using a response pattern of graded scores. Psychometr[illegible]a Monograph Supplement, 1969, 34 (4, Pt. 2, No. 17).

Samejima, F. Normal ogive model on the continuous response level in the multidimensional latent space. Psychometrika, 1974, 39, 111-121.

Samejima, F. A new family of models for the multiple-choice item (Research Report 79-4). Knoxville, TN: University of Tennessee, Department of Psychology, December 1979.

Samejima, F. Constant information model on the dichotomous response level. In D. J. Weiss (Ed.), Proceedings of the 1979 computerized adaptive testing conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, September 1980.

Shuford, E. H., Albert, A., & Massengill, H. E. Admissible probability measurement procedures. Psychometrika, 1966, 31, 125-145.

SILTS User's Manual. Princeton, NJ: Educational Testing Service (Mimeo).

Steffen, D. A., Gray, G. C., Wasmundt, K. C., & Lamos, J. P. Development of a low-cost, stand-alone microterminal for support of testing and instruction. Lowry Air Force Base, CO: Air Force Human Resources Laboratory, September 1978.

Sympson, J. B. A model for testing with multidimensional items. In D. J. Weiss (Ed.), Proceedings of the 1977 computerized adaptive testing conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, July 1978.

Vale, C. D. Computerized administration of free-response items. In D. J. Weiss (Ed.), Proceedings of the 1977 computerized adaptive testing conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, July 1978.

Vale, C. D. Development and evaluation of an adaptive testing strategy for use in multidimensional interest assessment. Unpublished doctoral dissertation, University of Minnesota, December 1980.

Vale, C. D. Design and implementation of a microcomputer-based adaptive testing system. Behavior Research Methods and Instrumentation, 1981, 13, 399-406.

Vale, C. D., & Weiss, D. J. A rapid search procedure to select items for a Bayesian adaptive test (Research Report 77-4). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, April 1977.

Vale, C. D., & Weiss, D. J. The stratified adaptive ability test as a tool for personnel selection and placement. TIMS Studies in the Management Sciences, 1978, 8, 135-151.

Wald, A. Sequential analysis. New York: Wiley, 1947.

Webster, T. Microcomputer Buyer's Guide. Los Angeles, CA: Computer Reference Guide, 1981.

Weiss, D. J. Strategies of adaptive ability measurement (Research Report 74-5). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, November 1974.

AD-A118 822 ASSESSMENT SYSTEMS CORP ST PAUL MN F/G 9/2
DEVELOPMENT OF A MICROCOMPUTER-BASED ADAPTIVE TESTING SYSTEM. P--ETC(U)
JUN 82 C D VALE, C ALBING, L FOOTE-LENNOX N00014-82-C-0132
UNCLASSIFIED ONR-ASC-82-01 NL

2 of 2

END
DATE
FILMED
9 82
DTIC

Weiss, D. J., & Betz, N. E. Ability measurement: conventional or adaptive? (Research Report 73-1). Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, 1973.

Wesman, A. G. Writing the test item. In R. L. Thorndike (Ed.), Educational Measurement. Washington, DC: American Council on Education, 1971.

Whitely, S. E. Measuring aptitude processes with multicomponent latent trait models. Journal of Educational Measurement, 1981, 18, 67-84.

APPENDIX A. DRAFT USER'S MANUAL

This appendix contains a draft of the User's Manual for the proposed system. This draft was sent to four experts in the computerized testing field for review. They were asked to comment on the system described, suggesting additions or deletions, and to comment on the clarity of the User's Manual, suggesting possible changes. Three of the experts provided critiques in time for inclusion in this report. The following manual is essentially the same version as was sent to the reviewers. Their comments will be incorporated into the final version.

The reviewers provided general and specific suggestions. The general consensus was that the system proposed would be very useful for both research and operational implementations of adaptive testing. Many specific suggestions were given. Some of the more substantial ones are listed below:

1. One reviewer felt that menu systems are much easier to use than systems requiring commands and languages. Specifically, he suggested that all operating-system commands be entered from a menu.

2. One reviewer felt that the instructions were at too high a level and that more basic instructions such as how to insert the discs and what a carriage return is should be included.

3. A status line at the bottom of the item editor was suggested as a useful feature in helping the user remember the edit commands.

4. The User's Manual did not succeed in fully defining all concepts and terms before they were used. The inclusion of a glossary was suggested.

5. Security and passwords were not discussed in the User's Manual. Since the system provides them, they should be discussed.

6. Several additions to the Chapter 2 bibliography were suggested.

7. The description of the graphics capabilities of the item editor was thought to be confusing and in need of expansion.

8. Two reviewers felt that the ARCHIVE facility provided too many opportunities for the user to destroy or misinterpret data. Several suggestions for making it foolproof were given.

9. One reviewer suggested that the item calibration routine should have item-parameter linking capabilities.

The reviews provided useful information for refining the manual. Where feasible, these suggestions will be incorporated into the final design and documentation of the system.

COMPUTERIZED ADAPTIVE TESTING SYSTEM

USER'S MANUAL

-- DRAFT --
Do Not Cite or Distribute

This document was prepared under
Contract N00014-82-C-0132 with the
Office of Naval Research

Assessment Systems Corporation
2395 University Avenue, Suite 306
St. Paul, MN 55114

Preface

This manual describes a proposed microcomputer-based adaptive test (CAT) development and administration system. In its current state, this manual is considered a working document for the design phase of the system; the prototype system has not yet been developed. The primary function of this document is to provide a description of the proposed system to potential users of the system so that they can suggest additional features and suggest deletion of features that will never be used. This document will, eventually, be developed into an actual user's guide for the prototype and final systems. Thus, it serves a second purpose in its current state in that potential users can, in addition to reacting to the proposed system, react to the presentation of the system. These reactions will be helpful in refining the final manual so that it can be easily understood by the typical user of the system.

All information necessary to administer, create, and analyze tests using the system is contained in this document, including how to enter new items and how to specify new testing strategies. Since the prototype system has not yet been built, some of the hardware descriptions are somewhat vague. This should not interfere with the integrity of the manual.

It is assumed that the reader is familiar with the basics of testing and statistics. A brief introduction to CAT is provided, but this is not intended to thoroughly familiarize the reader with the technology. No previous computer experience is necessary or assumed; such experience is undoubtedly helpful, however.

This manual is arranged by graduated introduction--no major concept is used before it has been defined and explained. Sections may be skipped if the information is not helpful to the reader but, in general, the manual should be read sequentially.

TABLE OF CONTENTS

CHAPTER 1. SYSTEM OVERVIEW

1.1. Introduction

One of the major advances in the field of psychological testing during the past thirty years is computerized adaptive testing (CAT). CAT offers a means of providing more efficient, less threatening human assessment than was possible with conventional tests. Using a CAT strategy, test items are tailored to the ability of each individual taking the test: each examinee receives the subset of test items most appropriate for his/her personal level of ability. (The term "ability" is used throughout this manual to refer to the examinee's trait level. The methods are general, however, and can be used for non-ability traits as well.) Measurement is thus more efficient, because time is not wasted with items of inappropriate difficulty, and less threatening, because items that are too difficult for the examinee are not administered.

Although CAT technology has been available for at least a decade, only recently has the cost of computerized delivery systems become low enough to make the implementation of CAT cost effective when compared to conventional paper-and-pencil test administration. The computer software to implement the technology still remains formidable, however, and cost-effective implementation can rarely be achieved when the software must be custom developed for a specific application.

This manual describes a turnkey adaptive testing system consisting of a computer and an adaptive-testing software package. This system allows a user to immediately begin development and administration of adaptive tests without the costs and delays of system development.

1.2. System Hardware +++to be supplied+++

1.3. System Software

The Computerized Adaptive Testing Language (CATL) software system eliminates most of the cost of software development in CAT implementation by providing a package of programs that can administer tests using common CAT strategies, and which can create other strategies through a simple authoring language. The CATL system consists of five subsystems: Test Administration, Item Banking, Test Construction, Item and Test Analysis, and Score Reporting and Interpretation.

The Test Administration subsystem, described in Chapter 3, administers the tests specified using other subsystems. It selects the items, presents them, calculates test scores, and (with optional hardware and software) monitors the performance of the examinee to detect problems and allow a test proctor to correct them during administration.

The Item Banking subsystem, described in Chapter 4, provides procedures through which test items, including the item text and relevent item characteristics, can be entered into a random-access database and can be refined and updated after they are entered.

The Test Construction subsystem, described in Chapter 5, provides a mechanism through which specific test items can be added to a strategy template (i.e., a strategy without items) provided either as part of the system or developed by the user. This subsystem also includes a powerful authoring language for developing new strategies, described in Chapter 8. This authoring language is the basic mechanism with which all tests are specified, including the ones provided with the system. Using this language, new testing strategies can be developed. Once the strategy is specified, many tests based on the strategy can be specified using the Test Construction subsystem.

The Item and Test Analysis subsystem, described in Chapter 6, uses the data collected during test administration to calibrate test items. It also provides estimates of test reliability and other indices of the psychometric performance of the test. The Test Evaluation subsystem, described in Chapter 7, uses item characteristics computed from previous administrations to allow the test developer to evaluate various testing methods and item sets for a specific application. Good estimates of test performance can thus be obtained without actually administering the test, and deficiencies can be corrected before valuable time is wasted on an unacceptable test.

The Score Reporting and Interpretation subsystem, described in Chapter 9, consists of an information retrieval system that can extract data from the Test Administration subsystem and allows the user to create score report forms and narrative interpretations of test-score data.

1.4 Summary

The system hardware and software combine to give the user of the computerized adaptive testing a powerful tool for test administration and scoring. Even more versatile are the features which allow the researcher to design and implement new testing strategies without having to program them in a standard programming language. Details of these procedures are discussed in succeeding chapters.

CHAPTER 2. AN INTRODUCTION TO COMPUTERIZED ADAPTIVE TESTING

2.1. Introduction

Administering a typical psychological test consists of presenting a sequence of test items and computing a test score. A testing "strategy" is defined by the methods used to select the test items and the procedure used to compute the test score.

There are two kinds of tests or testing strategies: conventional and adaptive. A conventional test is constructed by selecting a fixed set of items for administration to a group of individuals. A conventional test is typically scored by counting the number of items, from the fixed sequence, that are answered correctly. Conventional tests are simple to design and simple to score. They are not particularly efficient, however, because the same set of items is administered to the entire group. Conventional tests are efficient only if everyone in the group has the same level of ability. If this were the case, however, there would be no point in testing.

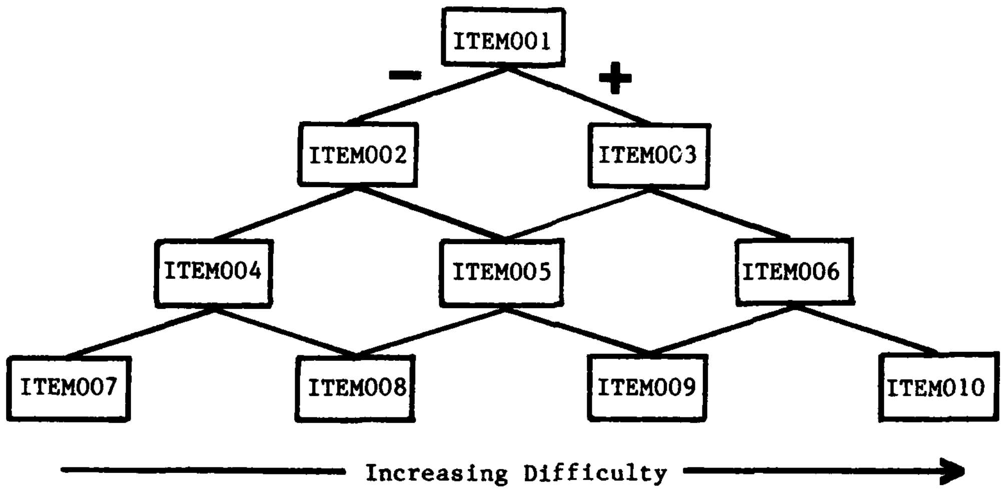
The concept of an adaptive test is quite simple: more information can be obtained from a test item if the item is matched to the ability level of the examinee. Low-ability examinees should get easy items; high-ability examinees should get more difficult items. Practical complications of adaptive testing come from two sources: (1) an examinee's ability level must be known to choose the appropriate items, but this is not known at the time testing begins, and (2) when everyone answers a different set of items, the test cannot be scored by simply counting the number answered correctly.

2.2. Intuitively-Branched Strategies

The solution to the problem of selecting items without knowing the ability level has been approached intuitively from several directions. The simplest solution was to create a hierarchy of tests, to use scores from one test to estimate the examinee's ability level, and then to appropriately select subsequent tests on the basis of the score on the first one. One of the earliest of the hierarchical strategies was the two-stage strategy in which all examinees first responded to a common routing test. The score on this test was then used to assign each examinee to a second-stage measurement test. Responses to both tests were then considered in arriving at a score. A problem with the two-stage strategy was that errors in measurement at the first stage resulted in a misrouting to an inappropriate measurement test.

The problem of misrouting in the first stage led to the development of multistage tests in which a common routing test led to two or more second-stage routing tests which, in turn, led to multiple third-stage measurement tests. The addition of extra routing tests reached its logical limit when only one item remained in each subtest. The most popular example of this limiting case was a triangular-shaped item structure, called a pyramidal test. In a pyramidal test, everyone starts at a common item and branches to an easier item after each incorrect response and to a more difficult item after each correct response. A diagram of a pyramidal test is shown in Figure 2-1.

Figure 2-1. Diagram of a Pyramidal Test



As the number of items at each stage of testing becomes smaller, the multistage structure begins to make very inefficient use of its items. While a 15-item two-stage test might require 45 items in the pool, a 15-item pyramidal test requires 120. This led to the development of re-entrant strategies in which a subtest could be partially administered, exited, and returned to. The best example of this form is the stradaptive strategy in which several (e.g., nine) subtests (or strata), are declared, each containing items at a specified difficulty level. In this strategy, testing proceeds by administering an item in one stratum and branching to a more difficult stratum if it is answered correctly, or to a less difficult stratum if it is answered incorrectly. Whenever testing branches to a stratum, the next unadministered item in that stratum is administered. A diagram of a stradaptive test is shown in Figure 2-2.

Several such mechanical branching mechanisms were evaluated over the years but no clear winners, in the psychometric sense, emerged. Solutions to the second problem, that of test scoring, did produce some superior strategies, however. While a few of the strategies above could be scored in simple, meaningful ways, many could not. The general solution to the item selection and test scoring problems lay in the basic theory that provided much of the original motivation for adaptive test administration, item response theory (IRT).

Figure 2-2. Diagram of a Stradaptive Test

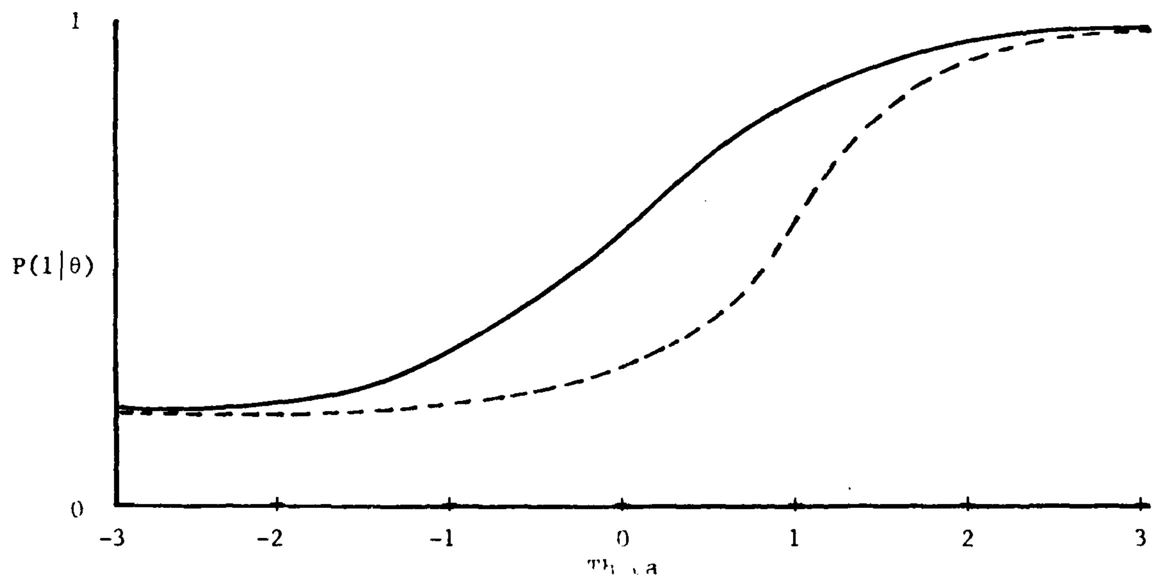
STRATUM A	STRATUM B	STRATUM C	STRATUM D	STRATUM E
ITEM-A1	ITEM-B1	ITEM-C1	ITEM-D1	ITEM-E1
ITEM-A2	ITEM-B2	ITEM-C2	ITEM-D2	ITEM-E2
ITEM-A3	ITEM-B3	ITEM-C3	ITEM-D3	ITEM-E3
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
ITEM-An	ITEM-Bn	ITEM-Cn	ITEM-Dn	ITEM-En
b = -2	b = -1	b = 0	b = 1	b = 2

2.3. Statistically-Branched Strategies

2.3.1. IRT models. IRT is a statistical theory consisting of a family of models that express the probability of observing a particular response to an item as a function of the characteristics of the item and the ability level of the examinee responding to the item. IRT models take on several forms, depending on the format of the item response and the simplifying assumptions made regarding the process underlying the response. The model that has been used most widely in CAT is the three-parameter logistic model. This model is applicable to multiple-choice questions scored in a dichotomous (e.g., correct-incorrect) manner. It describes the probability of observing a correct response to an item as a function of the examinee's ability level (theta) and three item parameters (*a*, *b*, and *c*). The *a* parameter indicates the test item's capacity to discriminate between levels of ability. It is related to the item-total correlation of a conventional test item. The *b* parameter is an index of the item's difficulty. It represents the level of theta (i.e., ability) at which the examinee has a 50-50 chance of knowing the correct answer to the item. The *c* parameter is equal to the probability that an examinee of very low ability could answer the item correctly by guessing. It is often referred to as the guessing parameter.

A three-parameter logistic item is mathematically described by its three parameters. These three parameters specify the item's operating characteristic curve (ICC). An example of two ICCs is shown in Figure 2-3. The ICC drawn with the solid line is for an item with *a* = 1.0, b = 0.0, and c = 0.2. The slope of the curve at any point is related to a; high values of a make it steeper in the middle. The location along the theta axis is a function of the b parameter; higher positive values shift the curve to the right. The lower left asymptote corrresponds to a probability or c of 0.2; high values of c raise the lower asymptote. The ICC shown with a dashed line is for an item with a = 2.0, b = 1.0, and c = 0.2. The midpoint of the curve has shifted to a theta of 1.0. The curve is steeper at points near this value. The lower asymptote remains at 0.2.

Figure 2-3. Item Characteristic Curves



The three-parameter model can be considered as a general model for dichotomously scored items. It allows items to differ in their discriminating power, their difficulty, and their resistance to guessing. This general form requires much computation, however, and if items can reasonably be assumed not to vary on all of these characteristics, computational savings can be obtained by eliminating some of the parameters.

The guessing parameter, c, causes the most difficulty and can be eliminated if the items cannot be answered correctly by guessing. Recall-type items that do not provide an opportunity for guessing can be used with the reduced model. The model with c assumed to be zero is called the two-parameter logistic model.

The discrimination parameter, a, can also be eliminated from the model if it is reasonable to assume that all items are equally good at discriminating high abilities from low abilities. This one-parameter model is usually referred to as the Rasch model, named after its developer.

Several IRT models expand upon the utility of the dichotomous-response models for other response formats. Polychotomous models, for instance, are appropriate where multiple response categories are scored on each item. One example of this situation is a multiple choice item in which the incorrect alternatives are weighted as a function of how "incorrect" they are. Another example is an interest-test item with ordered like, indifferent, and dislike responses. Still another is a performance rating scale on which performance is rated in ordered categories.

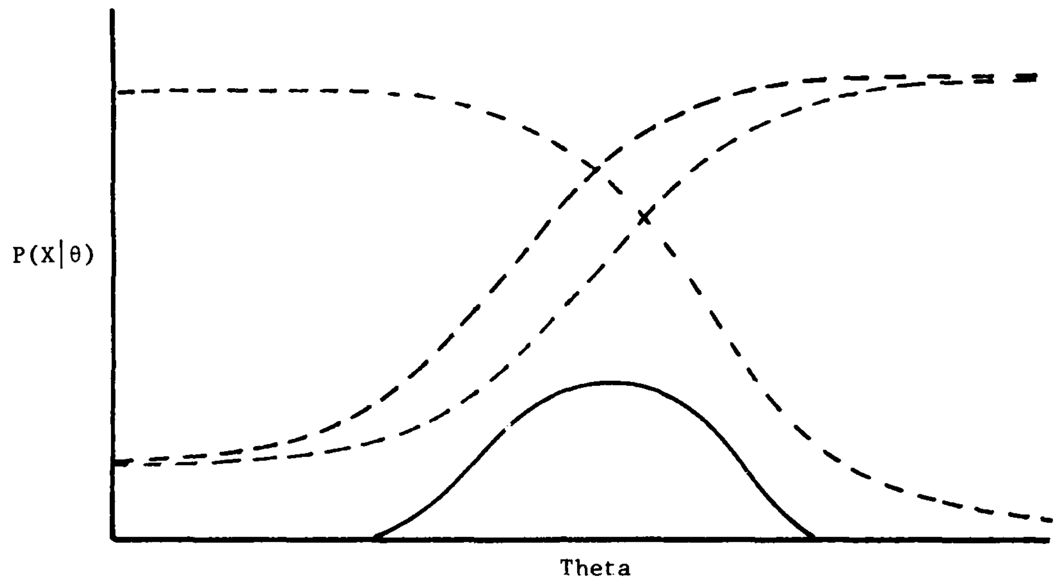
The general-case polychotomous models developed to date expand from the two-parameter model. Two basic forms exist: graded and nominal. The nominal model is the most general of the two. In effect, it scales each of the response options as a two-parameter item; each option has an _a_ and a _b_ parameter. The graded case is similar except that it assumes that all of the response categories within an item are equally discriminating. Thus a graded item has a _b_ parameter for each of its alternatives and one _a_ parameter.

Another family of IRT models assumes a slightly different shape of the ICC. All models discussed to this point have assumed that the response probabilities follow a logistic ogive (i.e., a specific shape of the response characteristics curve). Early in the development of IRT, several models were based on a normal ogive rather than a logistic ogive. The normal ogive model arose from the widespread use of the normal curve for statistical models in psychology. The shape of the ICCs is nearly the same for both models and it is difficult to say which better fits reality. The logistic ogive is more attractive, mathematically, because of its simplicity and has replaced the normal model in most practical implementations.

2.3.2. Scoring and general item selection methods. The ICCs shown in Figure 2-3 represent the probabilities of a correct response to each of the items. Complementary curves to each of these exist, representing the probability of an incorrect response. IRT allows the curves corresponding to the correct and incorrect responses in an examinee's response pattern to be multiplied together to yield a likelihood function. The likelihood function indicates the probability of observing the entire vector of responses at each level of ability. From this likelihood function, an estimate of the examinee's ability can be obtained. Conceptually, this can be done by assuming that the best estimate of an examinee's ability is the level of ability that would most likely produce the vector of responses observed. This score is called the maximum-likelihood estimate of ability.

Figure 2-4 shows three response characteristic curves, in dotted lines, for three items, two answered correctly and one answered incorrectly. The solid curve shows the product of these curves multiplied together (i.e., the likelihood function). The point at which the likelihood function reaches its peak is at a theta value of -------. The value of -------- is thus taken as the maximum-likelihood estimate of the examinee's ability.

Figure 2-4. A Likelihood Function With Two Items Correct



A problem sometimes encountered with maximum-likelihood estimation is that the procedure occasionally produces estimates of ability at positive or negative infinity on the theta scale. Rarely are ability levels of real examinees that extreme. An example of this is shown in Figure 2-5. Again, three items were presented. This time, however, the examinee answered all three correctly. The likelihood function has no obvious peak since the curve continues to rise as it leaves the right side of the figure. The maximum-likelihood estimate is positive infinity.

Several methods of bounding maximum-likelihood estimates have been used, many of them practical but arbitrary. An alternative to the maximum-likelihood method is the modal-Bayesian method. A modal-Bayesian estimate is, conceptually, very similar to the maximum-likelihood estimate. In fact, it is simply an extension of the maximum-likelihood estimate. It differs in that a Bayesian prior likelihood function is included when the response characteristics are multiplied together. This can have the effect of eliminating the infinite estimates. Figure 2-6 shows the effect, on the responses used in Figure 2-5, of assuming the distribution of theta, and thus the prior likelihood, to be standard normal. A finite peak of the function now exists at a theta value of -------.

Maximum-likelihood and modal-Bayesian ability estimation can be used for any item-selection strategy when the items are calibrated according to an IRT model. These scoring methods have suggested some new and improved item-selection strategies. Intuitively, it can be seen that the more peaked the likelihood function, the more accurate the estimate of ability. It thus makes good sense to explicitly select test items that will sharpen

Figure 2-5. A Likelihood Function With All Items Correct

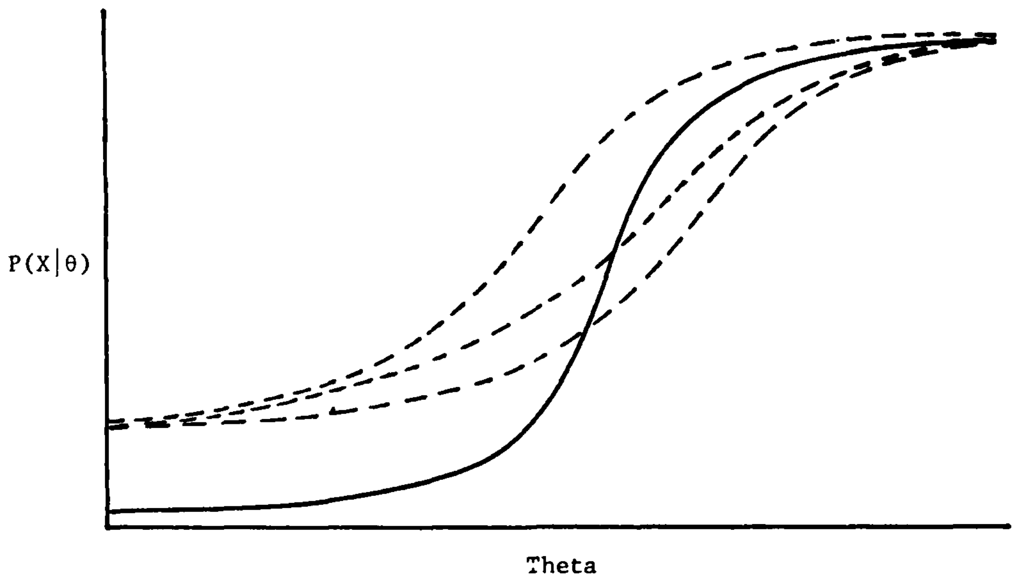
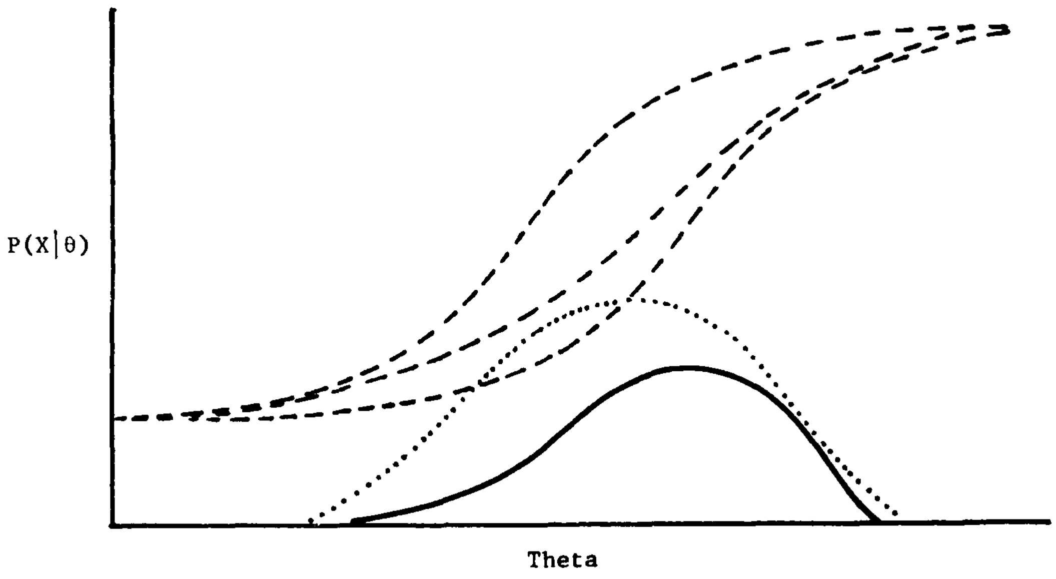


Figure 2-6. A Bayesian Posterior Likelihood Function With All Items Correct



the peak of the function. Although the ideal Bayesian and maximum-likelihood selection strategies differ slightly, very good item selection can be accomplished by selecting for both, using the maximum-likelihood ideal of selecting for maximum item information.

Item information is a statistical concept very closely related to the peakedness of the likelihood function and to the standard error of estimate of ability. Like the ICC, it is a function of theta and the parameters of the item. To select items based on it, an estimate of ability is first obtained. The information value for each item is then evaluated at that level of theta. The item with the highest value is chosen as the best item to administer. To form a test, a sequential process is specified in which an item is administered, an ability estimate is calculated,the item providing the most information at this estimate is selected, and the process is repeated. The sequential process may continue until a fixed number of items have been administered or until some other criterion for termination has been satisfied.

2.4 Summary

Computerized adaptive testing (CAT) is a method of testing in which the test items are tailored to an examinee's ability. A CAT strategy consists of a strategy for selecting items and a method for scoring the responses. Many CAT strategies have been developed and each may be useful in a particular environment. While this chapter does not replace understanding of and experience with CAT, it should provide some basic information on the field. The next chapter will cover more about how to operate the system itself.

Further information on CAT can be found in the following references:

Clark, C. L. (Ed.) Proceedings of the First Conference on Computerized Adaptive Testing. Washington, DC: U. S. Civil Service Commission, Personnel Research and Development Center, March 1976.

Lord, F. M. Applications of item response theory to practical testing problems. Hillsdale, NJ: Erlbaum, 1980.

Weiss, D. J. (Ed.) Proceedings of the 1977 Computerized Adaptive Testing Conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, July 1978.

Weiss, D. J. (Ed.) Proceedings of the 1979 Computerized Adaptive Testing Conference. Minneapolis: University of Minnesota, Department of Psychology, Psychometric Methods Program, September 1980.

CHAPTER 3. HOW TO TURN ON THE SYSTEM AND ADMINISTER A SAMPLE TEST

3.1. Turning On the System

Running a sample test is a good way to become familiar with the system. Three sample tests have been included in the system for you to administer and to study. This chapter contains the instructions you will need to administer these sample tests. After running the introductory tests, you will be ready to proceed to Chapters 4 and 5 which describe how to enter test items and how to revise the sample tests to include your own items. Chapter 5 will also describe how to create tests using other pre-defined testing strategies provided with the system.

You should begin by turning on the power to the system. To do this, insert the key in the slot and turn it one-quarter turn clockwise. The "Ready" light should come on and the terminal screen should display the message:

```
Insert disc and press B
```

Insert the disc marked "CATL Testing System — Main Disc" and press the B key on the keyboard. The disc will click and the following message will appear on the screen:

```
CATL testing system — Version X-X
:
```

The system is now running and ready for you to tell it what to do.

3.2. Administering Sample Tests

As you recall from Chapter 1, there are basically three types of tests: sample tests (in which everything is specified and ready to go), tests with pre-defined strategies (you provide only the items to be administered), and tests which you specify entirely (you write the strategy in CATL and provide the items). Three sample tests have been supplied as examples. These tests are titled SAMCONV (conventional-sample), SAMPYR (pyramidal-sample), and SAMSTRAD (stradaptive-sample). To run a sample test, remove the Main Disc and load the disc marked "Sample Tests (Chapter 3)."

The general command to see the names of the tests on the disk is SHOW. If you enter the command by itself, it will provide a list of everything on the disc. To get a list of only the tests on the disc, you should enter the command:

```
SHOW *.TST
```

To get a list of only the sample tests on the disc, you should enter the command:

```
SHOW SAM*.TST
```

If you enter the latter command, the list in Figure 3-1 should be displayed:

Figure 3-1. Display of Sample Tests

```
SAMCONV  - CONVENTIONAL TEST (10 ITEMS): MATH
SAMPYR   - PYRAMIDAL (20 ITEMS): HISTORY
SAMSTRAD - STRADAPTIVE TEST (20 ITEMS): VOCABULARY
```

To administer one of the tests, enter the command:

```
ADMIN
```

This command will cause the test administration program to run. The first thing it will ask for is the name of the test to run. The example in Figure 3-2 shows the administration of the sample pyramidal test. The parts that you have to type have been underlined. The symbol <CR> is used to indicate a carriage return.

Figure 3-2. Administration of a Pyramidal Test

```
: ADMIN <CR>

   Test Name? PYRAMID <CR>
   Date?  14 FEBRUARY 1982 <CR>
   Who is taking this test?  A. C. CLONES <CR>
   Notes?  SECOND TEST FOR THIS EXAMINEE; FIRST TEST INVALID <CR>

   Press return to start the test   <CR>
```

A seven-item history test will be administered. At the end of the test, the test score will be displayed.

3.3 Summary

To run the testing system, a system disc is inserted in the right-hand disc drive and the "b" on the keyboard is pressed. Test names are listed by typing SHOW *.TST and the tests are administered by typing ADMIN. A sample test was demonstrated in this chapter. The next chapter begins to describe how to create tests of your own.

CHAPTER 4. ENTERING AND EDITING ITEMS

4.1. Introduction

Psychological tests are composed of test items. To administer any of the computerized tests (other than the sample tests), items must first be entered into a computerized item bank. Typically, items need to be modified after they are first administered and these changes also need to be entered into the item bank. The item editor, EDIT, is the system program that allows you to do this.

EDIT is a CRT screen-oriented editor. It displays an entire item on the CRT screen, both item text and item characteristics, and allows you to make whatever changes you desire. An existing item can be called up and modified with EDIT. New items can be created by "editing" a blank item.

This chapter begins by describing the item formats available in the system. Then, the procedure for creating a new item is described. Finally, these procedures are applied to the editing of an existing item. When you have completed this chapter, you should be able to enter new test items and modify existing ones.

4.2. Test Item Formats

4.2.1 Textual items. A textual item is a test question which consists only of characters; it does not include pictures or drawings. Three types of textual items are handled by the testing system: dichotomous, graded, and nominal. In the editor, all formats display the item characteristics at the top of the screen and the item itself at the bottom. The difference among the formats is in the item-characteristic section. The dichotomous item format is shown in Figure 4-1.

The item characteristics are defined as:

Number	--	The item number
Nalt	--	The number of alternatives
Disp-Time	--	The time in tenths of seconds to display the text (-1 for untimed display)
Resp-Time	--	The time in tenths of seconds allowed for a response after the display has disappeared (-1 for unlimited time)
Clear	--	Clear the screen before presenting the stimulus? (Y/N)
Key	--	The correct (keyed) response
A	--	The IRT $\underline{a}$ parameter
B	--	The IRT $\underline{b}$ parameter
C	--	The IRT $\underline{c}$ parameter

Figure 4-1. Dichotomous Item Screen Display

```
Number: ______  Nalt: _ Disp-Time: ____  Resp-Time: ____  Clear: _  Key: _
A: ______  B: ______  C: ______

 1:____________________________________________________________________
 2:____________________________________________________________________
 3:____________________________________________________________________
 4:____________________________________________________________________
 5:____________________________________________________________________
 6:____________________________________________________________________
 7:____________________________________________________________________
 8:____________________________________________________________________
 9:____________________________________________________________________
10:____________________________________________________________________
11:____________________________________________________________________
12:____________________________________________________________________
13:____________________________________________________________________
14:____________________________________________________________________
15:____________________________________________________________________
16:____________________________________________________________________
17:____________________________________________________________________
18:____________________________________________________________________
19:____________________________________________________________________
20:____________________________________________________________________
```

Figure 4-2. Dichotomous Test Item

```
Number: DEM001 Nalt: 5 Disp-Time: 6000 Resp-Time:    0 Clear: Y Key: A
A:  1.322 B: -0.301 C:  0.214

 1:What kind of test administration is most efficient?
 2:____________________________________________________________________
 3:     A. Adaptive
 4:     B: Conventional
 5:     C: Random
 6:     D: Abstract
 7:     E: Cyclic
 8:____________________________________________________________________
 9:____________________________________________________________________
10:____________________________________________________________________
11:____________________________________________________________________
12:____________________________________________________________________
13:____________________________________________________________________
14:____________________________________________________________________
15:____________________________________________________________________
16:____________________________________________________________________
17:____________________________________________________________________
18:____________________________________________________________________
19:____________________________________________________________________
20:____________________________________________________________________
```

The text of the item appears on the 20 lines following the characteristics in exactly the format it will be presented on the screen. The first column on the screen is the column immediately following the colon. An example of what a completed item might look like is shown in Figure 4-2.

This item, labeled DEM001, has five alternatives. The item-text display will be presented for a maximum of 10 minutes or until the examinee responds. The examinee will be given no time to respond after the display disappears. The screen will be cleared before the text is presented. The correct response is "A." The *a*, *b*, and *c* parameters are 1.322, -0.301, and 0.214.

A graded item has a similar format but the parameter line is different, as shown in Figure 4-3.

Figure 4-3. Graded Item Characteristics

```
Number: DEM002 Nalt: 4 Disp-Time: 6000 Resp-Time:    0 Clear: Y Key: C
A:   1.322 B1: -0.544 B2:  0.223 B3:  0.978 B4: ______ B5: ______
```

A graded item has one *a* parameter and, in this system, up to five *b* parameters. In the example above, the item has four alternatives and thus three *b* parameters. (The *b* parameter in a graded item refers to the boundary between two response categories.)

A nominal item has two lines of parameters, one *a* parameter and one *b* parameter, for each response category. The item characteristic lines for a nominal item look like the sample in Figure 4-4.

Figure 4-4. Nominal Item Characteristics

```
Number: DEM003 Nalt: 4 Disp-Time: 6000 Resp-Time:    0 Clear: Y Key: C
A1:  1.021 A2:  0.978 A3:  1.302 A4:  0.828 A5: ______ A6: ______
B1: -1.222 B2: -0.454 B3:  1.123 B4:  0.802 B5: ______ B6: ______
```

In this example the item has four alternatives, each with corresponding *a* and *b* parameters.

4.2.2 Graphic Items. Graphic items may have any of the three formats available to the textual items. A graphic item is displayed in the same general format except that the textual area of the display is replaced by a graphic display. As will be discussed in the sections that follow, only the item characteristics of a graphic item may be edited with EDIT and special procedures are used to enter the graphic portions.

4.3. Creating a New Item

To start the EDIT program, first enter the command:

```
     EDIT <CR>
```

The system will respond with:

```
     EDIT -- Test Item Editor -- Version X.X

     Enter an Item Number, a Format, or QUIT: _________
```

If you enter an item number at this point, the item will be fetched and displayed. To enter a new item, a format should be specified. EDIT recognizes items in three formats: dichotomous, graded, and nominal. The three corresponding titles are DICHOT, GRADED, and NOMIN. If any one of these is entered, a blank item format will appear on the screen. This format can then be filled in using the operations described below.

4.3.1. General editing operations.

Six editing operations are available in EDIT: Change, Insert, Delete, Save, Abort, and Quit. When the item format appears, you are automatically in the Change mode. The other modes can be entered by holding down the control (ctrl) key on the terminal and simultaneously typing I for Insert, D for Delete, S for Save, A for Abort, or Q for Quit. Change mode can be re-entered by holding the control key and typing C.

The cursor (the white square on the screen) can be moved to any location where changes or additions are to be made. You can move the cursor around the screen with the four arrow keys at the upper left of the keyboard.

In Change mode, any character typed will replace the character under the cursor. In this mode, you can make character-by-character changes in the text or item characteristics by simply typing over whatever characters presently exist. Formatting characters (e.g., the characteristic labels) are protected and typing over them will not change anything.

In Insert mode, characters are inserted into a line directly to the left of the cursor. Characters to the left of the cursor can be deleted by backspacing over them. Characters to the right of the cursor can be deleted by moving the cursor to the right of them and then backspacing over them. A new line can be inserted into the text by pressing the return key while in Insert mode. The new line will appear immediately following the cursor and the cursor will move to the beginning of that line.

Delete mode is a temporary mode. When control-D is typed, the line containing the cursor is deleted. The mode is then automatically returned to the previous one. If, for example, you were in Insert mode and typed control-D, the line under the cursor would disappear and EDIT would return to Insert mode. Typing control-D again would make another line disappear.

Control-P causes the editor to make the changes permanent. The item remains on the screen for further editing. The item made permanent is determined by the item number set on the screen. If the item number is changed from the original, the item will create or replace the item of that number. Multiple copies of the item can be made by simply changing the item number and typing control-P again.

Control-A clears the screen and asks for another item without making any of the changes permanent.

Control-Q causes the editor to make the changes permanent, clear the screen, and ask for another item. You can terminate the editor at this point by entering QUIT instead of an item format or an item number.

4.3.2. Graphic items. A graphic item is entered through the system digitizer port rather than through the terminal. To enter a graphic item, begin the process like a textual item. When you are ready to enter the item test, type control-G. This tells EDIT to read the text from the system digitizer port and identifies the item as a graphic item. The item will be displayed on the screen as the information is received from the digitizer. During the digitizing process, the cursor will disappear and you will not be able to execute any edit commands other than control-A to abort the process. When the process is complete, the cursor will reappear at the upper left of the screen. The EDIT operations will once again be available.

4.4. Editing an Existing Item

An existing item can be edited in a manner very similar to the way a new item is entered. Instead of specifying a format when EDIT asks for an item number or format, you simply enter the number of the existing item. This item appears on the screen with the appropriate format and you can edit it using the basic edit operations described above.

You will not be able to edit the graphic portion of a graphic item. You can, however, redigitize the graphic portion of the item without altering the characteristic portion. To do this, simply enter control-G and proceed with the digitization as before.

4.5 Summary

Items form the core of psychological testing. To use any of the tests subsequently described you must have a bank from which to draw items. The system facilitates item entry, editing, and retrieval, making it possible to maintain and change a large bank of items.

CHAPTER 5. DEVELOPING A TEST FROM A TEMPLATE

5.1 Introduction

Recall that a test is a collection of items. To administer a test, you must first have a pool of test items. A strategy is a particular collection of items sequenced in a particular way. Strategies can be very simple (such as a list of items that are administered sequentially) or quite complicated (a stradaptive branching strategy, for example).

Several testing strategies have been provided with the system. These strategies are provided as test templates. A test template is a general test specification in an author language (discussed in Chapter 8) that has several blanks in it allowing you to fill in the items to administer as well as certain test characteristics. The templates provide an administration framework, allowing you to construct tests by simply specifying the items to be used. You can use items in the database for any of these pre-defined strategies.

5.2 Specifying Items

To create a test from a template, you need to do two things. First you need to select and fetch a template. Then you need to fill in the data required by the template.

A list of the templates provided with the system along with descriptions of them can be found in Appendix (4). Each description details the testing strategy. For example,

> FIXLENCN. This template contains a fixed-length conventional test. You will be asked to supply a test title and as many test items as you wish to include.

In order to use any of the templates, first enter the command "SPECIFY." The system will respond

> "Would you like to see a brief list of template titles available? Answer y/n/help."

If you answer Y, the system will display a list of the templates available. (Again, to see a complete list of the template descriptions, refer to Appendix (4).) At the end of the list, the system will say

> "Please select the template you want to specify."

If you answer N, the system will respond

"Please select the template you want to specify."

without giving you a list of the available templates. HELP will result in an explanation of the question and directions to refer to the manual for more details.

After you enter the name of the template you want, the system will ask

"Is this the template you wish to specify?"

If you have specified the wrong template, you can answer N at this time and enter the correct template name. Otherwise, a Y response will begin the processing of the template.

The template directs construction of the test. The system will ask you for a name for the test. A mnemonic (perhaps including a date or version number of the test you are about to create) is usually most appropriate. Next, the system will typically ask for item numbers, specifying their sequence automatically.

Each template is different; some are more complicated than others. For illustration, the sequence for specifying a fixed length conventional test is shown in Figure 5-1. Your responses are underlined and capitalized.

This particular specification of the template "FXDLENCN" will contain items ITM102, ITM209, ITM210, ITM211, and ITM329. It will reside in memory as a complete test specification and will always be referenced with the name "FXDVER1.SPC."

A variable length conventional test requires a more complicated template. Figure 5-2 shows the specification of a variable-length conventional test. The primary difference between the variable-length and fixed-length tests is that the variable-length test has a logical termination criterion. Again, your responses are underlined and capitalized.

5.3 Consolidating a Test

When you have completed specification of the items, your test will exist in two or more parts. The test specification will be on the file created by SPECIFY and the items themselves will be in one or more banks. If your system has small discs, the item banks may be on several different discs. To make tests execute efficiently, it is helpful to consolidate all of the test information on one file and to do as much preprocessing as possible. This is done by the program CONSOLIDATE.

Figure 5-1. Specification of a Fixed-Length Conventional Test

```
SPECIFY <CR>

Would you like to see a brief list of template titles available?

:Y <CR>

VARLENCN -- conventional test with variable termination
FXDLENCN -- conventional test (fixed length)
PYRAMIDL -- pyramidal testing strategy
BAYESIAN -- Bayesian testing strategy
          o
          o   etc.
          o
Please select the template you want to specify.

:FXDLENCN <CR>

Conventional fixed length test -- Is this the template you wish to specify?

:Y <CR>

Conventional, fixed length test.  Enter test title.

:FXDVER1.SPC <CR>

Enter test item numbers, one at a time, using an additional
carriage return as termination.

#ITM102 <CR>
#ITM209 <CR>
#ITM210 <CR>
#ITM211 <CR>
#ITM329 <CR>
# <CR>

Enter the test name again for verification.

:FXDVER1.SPC <CR>
```

Figure 5-2. Specification of a Variable-Length Conventional Test

```
SPECIFY ITEMS <CR>

Would you like to see a brief list of template titles available?

:N <CR>

Please select the template you want to specify.

:VARLENCN <CR>

Variable length conventional test template -- Is this the
template you wish to specify?

:Y <CR>

Variable length conventional test template -- Enter test title

:VARLVER1.SPC <CR>

Enter test item numbers, one at a time.  When you have entered them all,
enter a carriage return alone.

#ITM602 <CR>
#ITM671 <CR>
#ITM772 <CR>
#ITM778 <CR>
#ITM792 <CR>
#ITM809 <CR>
#ITM851 <CR>
# <CR>

Enter the termination criterion enclosed in parentheses (scores available
are PCORR and FORMULASCORE).

:(PCORR < .3 OR PCORR > .8) <CR>

Enter the test name again for verification.

:VARLVER1.SPC <CR>
```

To run the consolidator, enter the command:

```
     CONSOLIDATE
```

The system will respond with:

```
     Test Consolidator Version X.X -- Process Test Specifications

     Enter the name of the test specification file: _______________
```

You should enter the name of the test you created using SPECIFY. If you name a test that does not exist, the consolidator will respond with:

```
     The test you named does not exist.
     Enter the name of the test specification file: _______________
```

At this point, you should either enter the proper file name or STOP to stop. When the consolidator finds the specification file, it will respond:

```
     Enter a name for the executable test file: _______________
```

You should enter the name that you would like the test to have. This is the name that you will use when you administer the test. If the file already exists, the consolidator will give you a warning:

```
     File XXXXXXXX.XXX already exists -- Can it be erased? (Y/N) __
```

If you respond with an N, you will be asked to rename the file. If you respond with a Y, the old file will be erased. When an acceptable file name has been entered, the consolidator will respond:

```
     Processing specification file XXXXXXXX.XXX
```

It will consolidate the specification and the items. If it requires an item bank that is not on a current disc, it will give the message:

```
     Item Bank XXXXXXXX is not available.  Insert proper disc and
     enter GO.
```

If you insert the proper disc and type GO, the consolidation will continue. If the proper bank does not exist, the consolidation cannot continue and you should enter STOP to end it.

When the consolidation is complete, the consolidator will issue the message:

```
     Consolidation complete -- The test is on file XXXXXXXX.XXX
```

At this point, the executable test file has been created and the test is ready to administer.

5.4 Summary

The templates provided can be used to specify a wide variety of tests using your own items and scoring criteria. They allow you to insert items into a skeleton strategy; you need not thoroughly understand the underlying computer concepts to create your own adaptive tests.

Templates provided with the system can be completed using the system program SPECIFY. This produces a complete test specification. Completed test specifications need to be consolidated before they are administered. This is done using the program CONSOLIDATE.

CHAPTER 6 — CALIBRATING TEST ITEMS

6.1. Introduction

Many useful adaptive testing strategies are based on item response theory (IRT), discussed in Chapter 2. To use IRT, the test items must be calibrated. That is, the parameters that describe the statistical behavior of the items as a function of ability must be estimated.

A calibration program is provided with the system to estimate parameters for the three-parameter logistic model. As described in Chapter 2, the $\underline{a}$ parameter describes the item's capacity to discriminate among different levels of ability, the $\underline{b}$ parameter describes the item's difficulty, and the $\underline{c}$ parameter is the probability that the item can be answered correctly by guessing. The $\underline{c}$ parameter is primarily useful for ability items where guessing is likely. For other item types, such as personality items, the $\underline{c}$ parameter can be set to zero since guessing is not a factor. Similarly, if there is reason to believe that all items are equally discriminating, the $\underline{a}$ parameters can all be set to a constant and the items will be calibrated according to a form of the one-parameter logistic or Rasch model.

There are four steps required to calibrate items: data collection, item calibration, interpretation, and item bank updating.

6.2. Collecting the Data

Item calibration usually begins with the administration of the items to a large number of examinees in the form of a conventional test. When the testing system administers a test to an individual, it can be made to keep item response data. (This is discussed in detail in Chapter 8.) It keeps the data for each individual on a separate file. Calibrations may use data from well over 1000 examinees, and storing data in this form is inefficient. A utility has been provided to condense the data into a more compact form. This utility is called ARCHIVE.

ARCHIVE condenses all data files with the file qualifier DTA. For example, the data files SB11376.DTA, SB48853.DTA, and SB97301.DTA will all be archived onto a specified archive file.

The individual examinee data files read by ARCHIVE must contain single conventional tests and all of the files to be archived must contain the same test. The template CALIBCON is provided to assist you in constructing such tests. Figure 6-1 shows an example of a test created using the template.

The test administration program will create a file with a file name composed of the examinee's identification and the appropriate file qualifier.

Figure 6-1 -- Specification of a Calibration Test

```
:SPECIFY <CR>

Would you like to see a brief list of template titles available?

:N <CR>

Please select the template you want to specify.

:CALIBCON <CR>

Calibration test -- Is this the template you wish to specify?

:Y <CR>

Calibration template -- Use to collect data for item calibration.

Enter the item numbers of the items to be included, one at a
time and finish by pressing the return key alone.

#TST035 <CR>
#TST198 <CR>
#TST132 <CR>
#TST004 <CR>

     o
     o
     o

# <CR>
```

To archive the data in all of the DTA files into a single file, the archive program can be run by entering the command:

```
ARCHIVE
```

ARCHIVE will respond with:

```
Archive version X.X
Enter the name of the archive file: _______________
```

When the archive file name is entered, ARCHIVE will collect data from all of the DTA files and write them to the end of the archive file. If the named archive file has not been created, ARCHIVE will create it. If it has been created and contains data from previous archive runs, the new data will be added to the end of it.

After running ARCHIVE, the original examinee files should be erased to conserve space. The system command for erasing all of the DTA files is:

```
ERA *.DTA
```

If the files are not erased, they should be renamed with a file qualifier other than DTA before ARCHIVE is run again. This is neccessary because ARCHIVE collects data from all DTA files. If ARCHIVE is run twice with files having the DTA qualifier still on the disc, the data will be archived twice and the archive file will contain duplicate data.

6.3. Calibrating the Items

The calibration program, CALIBRAT, is an interactive calibration program. It reads the input data from the archive file and writes the parameters to a parameter file. Control of the program, however, is from the terminal. CALIBRAT can be run by entering the command:

```
CALIBRAT
```

CALIBRAT will respond with:

```
Calibrat -- Three-parameter logistic item calibration -- Version X.X

What is the name of the input (archive) file? _______________
```

You should respond with the name of the file onto which the individual examinee files were archived. If the named file is nonexistent or empty, CALIBRAT will respond with an appropriate message and will again ask for the input file name. If you wish, you can abort the program by entering STOP. If the named file contains valid data, CALIBRAT will respond by asking for the name of the output file:

```
Name the file on which to write the item parameters: _______________
```

You should respond by entering the name of a file where you would like the item parameters to be written. If the file named already exists, CALIBRAT will ask if it is all right to erase it:

```
File XXXXXXXX.XXX already exists -- Can it be erased? (Y/N) __
```

If you enter Y, the file will be erased and CALIBRAT will continue. If you enter N, CALIBRAT will again ask you to name the output file. The final file needed by CALIBRAT is the file for the summary output:

```
Name the file on which to write the item-analysis summary: _________
```

You should respond by entering the name of a file where you would like the item analysis summary to be written. If the file named already exists, CALIBRAT will ask if it is all right to erase it:

```
     File XXXXXXXX.XXX already exists -- Can it be erased? (Y/N) __
```

If you enter Y, the file will be erased and CALIBRAT will continue. If you enter N, CALIBRAT will again ask you to name the file.

You can put bounds on each of the three item parameters. This feature can be used to keep item parameters within reasonable limits when the data suggest otherwise. It can also be used to fix any parameter for a restricted model. A parameter can be fixed by simply setting both the upper and lower bounds to the same value. CALIBRAT always asks for bounds for the parameters. If you do not wish to bound the parameters, simply press the return key after each request without entering anything. To get the bounds, CALIBRAT will ask:

```
     Enter the upper and lower bounds for the "a" parameter
     separated by a comma: ________________
```

If two values are entered and the second is greater than or equal to the first, those values will become the parameter bounds. If only a return is entered, the default bounds of 0.0 and 2.5 will be used. If anything else is entered, CALIBRAT will refuse to accept it and will repeat the request. When the a parameter bounds are set, the b parameter bounds will be requested:

```
     Enter the upper and lower bounds for the "b" parameter
     separated by a comma: ________________
```

If two values are entered and the second is greater than or equal to the first, those values will become the parameter bounds. If only a return is entered, the default bounds of -2.5 and 2.5 will be used. If anything else is entered, CALIBRAT will refuse to accept it and will repeat the request. When the b parameter bounds are set, the c parameter bounds will be requested:

```
     Enter the upper and lower bounds for the "c" parameter
     separated by a comma: ________________
```

If two values are entered and the second is greater than or equal to the first, those values will become the parameter bounds. If only a return is entered, the default bounds of 0.0 and 0.5 will be used. If anything else is entered, CALIBRAT will refuse to accept it and will repeat the request. To set a parameter to a constant value, the lower and upper bounds must both be set equal to that value.

CALIBRAT will then begin to calibrate the items. During the process of calibration, CALIBRAT will provide data to the terminal to indicate the status of the calibration run. CALIBRAT makes several passes through the data, refining the parameter estimates on each pass. At the end of each pass, CALIBRAT will display the number of the current pass, the time required to complete that pass, and after the third pass, the estimated number of passes and estimated time to completion. The terminal display of a calibration run might look like the one in Figure 6-2.

Figure 6-2. A Sample Calibration Run

```
:CALIBRAT <CR>

Calibrat -- Three-parameter logistic item calibration -- Version X.X

What is the name of the input (archive) file? ITEMDATA.ARC <CR>

Name the file on which to write the item parameters: ITEMPARS.CAL <CR>

File ITEMPARS.CAL already exists -- Can it be erased? (Y/N) Y <CR>

Name the file on which to write the item-analysis summary: TEMP.OUT <CR>

Enter the upper and lower bounds for the "a" parameter
separated by a comma: 0.6, 2.5 <CR>

Enter the upper and lower bounds for the "b" parameter
separated by a comma: <CR>

Enter the upper and lower bounds for the "c" parameter
separated by a comma: <CR>


          CURRENT STATUS           PROJECTED STATUS

           PASS   TIME                PASS   TIME
             1     47
             2     90
             3    125                   6     210
             4    153                   6     212
             5    184                   7     240
             6    205                   7     242
             7    238                   7     238

      Calibration of 50 items on 1350 examinees completed
```

In this example, calibration was completed in 238 seconds after making seven passes. The estimates of the time to completion improved as the calibration proceeded. A maximum of 25 stages is permitted by CALIBRAT. If the estimates have not converged to an acceptable accuracy by that time, CALIBRAT terminates, reports the current estimates, and provides a message that the process did not converge.

6.4. Interpreting the Analyses

Upon completion of the run, the parameter estimates are written to the appropriate output file. The file begins with a title line including the input file name and the date of the run. All data for each item are written on single lines in the file. Each line begins with the item reference number and continues with the three parameters _a_, _b_, and _c_, followed by their respective standard errors. An example of the first few lines of such a file is shown in Figure 6-3.

Figure 6-3. Parameter Output File

```
ITEM PARAMETERS FOR DATA ON FILE ITEMDATA.ARC -- 17 MAY 82
TST035  0.955  1.223  0.290  0.133  0.055  0.042
TST198  1.333  0.890  0.220  0.091  0.043  0.067
TST132  1.544 -0.678  0.325  0.078  0.056  0.136
TST004  1.233  2.500  0.122  0.099  1.335  0.088  *

                       o
                       o
                       o
```

You should look at this file to determine if problems were encountered in estimating any of the item parameters. If the standard error of a parameter is too large or if any of the parameters reached one of their bounds, an asterisk will appear to the right of the last standard error. The parameters of any item so marked should be checked to ensure that they are acceptable. If not, you may want to delete the parameters for the item so that they will not be transferred to the item bank.

Item TST004 provides an example of an item with estimation problems. Its _b_ parameter hit the upper bound at 2.5 and was estimated poorly, as evidenced by the standard error of 1.335. If this poor estimation is unacceptable, the line in the file containing the parameters should be deleted using the system editor.

An extended item analysis is written to the file named for the summary file. Both classical and IRT item and test analyses are provided on this file. An abbreviated listing of a sample of this file is shown in Figure 6-4.

Figure 6-4. Item and Test Analysis

```
CALIBRAT Version X.X  --  Item and test analysis

Input file : ITEMDATA.ARC     Date : 17 May 82

Number of examinees : 1350    Number of items : 50

Classical test analyses

     KR-20 reliability :          .873
     Mean (number correct) :    39.750
     Standard deviation :        6.300
     Standard error :            2.245


Item response theory analysis

     Mean a :                    1.237
     Mean b :                     .242
     Mean c :                     .189
     Total information :        XX.XXX
     Expected information :     XX.XXX
```

Test information curve

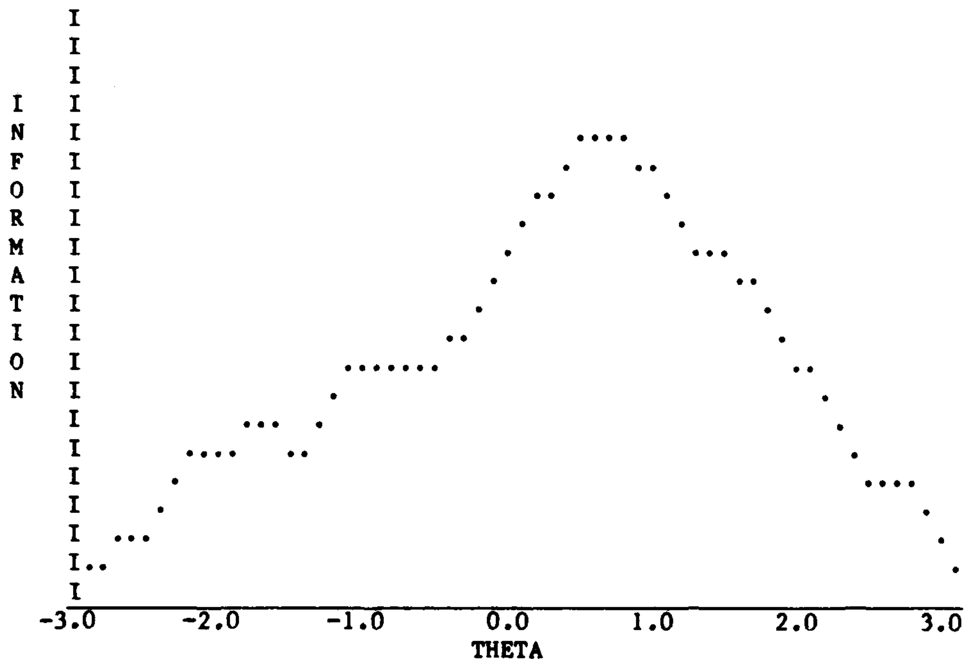


Figure 6-4 (Continued)

Item analysis

Item	a	b	c	Se(a)	Se(b)	Se(c)	P	Pbis	Check
TST035	0.955	1.223	0.290	0.133	0.055	0.042	0.455	0.513	
TST198	1.333	0.890	0.220	0.091	0.043	0.067	0.534	0.602	
TST132	1.544	-0.678	0.325	0.078	0.056	0.136	0.723	0.676	
TST004	1.233	2.500	0.122	0.099	1.335	0.088	0.278	0.799	*

o

o

o

As Figure 6-4 shows, the summary output is divided into four sections. The first section provides documentation for the run indicating the data file used, the date of the run, the number of examinees included, and the number of items calibrated.

The second section provides classical test analysis statistics. The KR-20 internal consistency reliability is computed along with the mean number-correct score, the standard deviation, and the standard error of measurement.

The third section presents IRT test analyses. First the means of the three IRT item parameters are presented. Then, the total-test information is provided. This corresponds to the total area under the test information curve. The expected information is the expected value of the test information curve weighted by a standard normal density function. The graphic plot is the total available test information plotted as a function of theta. It is evaluated at 61 equally spaced theta values between -3.0 and 3.0.

The final section presents the individual item characteristics, one line per item. The first column contains the item reference number. Columns two through four contain the three item parameter estimates. The next three columns contain the standard errors of the parameter estimates. Columns eight and nine contain the classical proportion passing or "p" value and the point-biserial item-total-score correlation. The final column may contain an asterisk if any of the parameters reached a bound or if the standard errors were too large. This is the same indicator that appears in the last column of the parameter file.

6.5. Updating the Item Bank

After the items have been calibrated, the parameters should be entered into the item bank for use in future test construction. There are two ways by which this can be done. The first is to manually take the data from a printout of the output from CALIBRAT and then enter it into the bank using the item editor. A much more efficient way is to use the utility, UPDATE.

UPDATE reads the CALIBRAT output file and transfers all of the parameters to the appropriate items in the bank. To run UPDATE, you should enter the command:

```
UPDATE
```

The system will respond with:

```
Update version X.X -- Transfer parameters to item bank

What is the name of the parameter file? ______________
```

You should enter the name of the parameter file. UPDATE will then store all of the parameters in the item bank along with the items. A log of the items updated will be displayed on the terminal. If for any reason the parameters cannot be added for an item, an error message will be displayed and the updating will continue with the next item. An example of an UPDATE run is shown in Figure 6-5.

Figure 6-5 -- A Sample Update Run

```
:UPDATE

Update version X.X -- Transfer parameters to item bank

What is the name of the parameter file? ITEMPARS.CAL

TST035  Updated
TST098  Updated
TST132  Item not in bank -- Not updated
TST004  Updated

         o
         o
         o
```

6.6 Summary

IRT item parameters are required for most adaptive testing methods. For items scored dichotomously (e.g. right/wrong), these parameters can be estimated using the facility CALIBRAT. Calibration requires four steps: (1) creation of a conventional test containing the experimental items, (2) estimation of the parameters, (3) evaluation of the results, and (4) insertion of the parameters into the item bank. When this has been accomplished, the items are ready for inclusion in an adaptive test.

CHAPTER 7. TEST PRE-EVALUATION

7.1. Introduction

One distinct advantage provided by item response theory (IRT) is the ability to predict the characteristics a test will have by using statistics from past administrations. Although the specific test may never have been administered before, it can be evaluated if the items have been calibrated.

The program provided to perform this evaluation is EVALUATE. EVALUATE works from the executable test file produced by the consolidator. It provides the following data:

Estimated test reliability

Estimated subtest reliability

Test information functions

Subtest information functions

Average item discriminating power

Average item difficulty

Average item guessing parameter

EVALUATE can be used with both conventional and adaptive tests. The information provided shows the test's characteristics in the limiting case where all items are to be administered.

7.2. Running EVALUATE

EVALUATE is run by entering the command:

```
EVALUATE
```

The system will respond with:

```
Evaluate -- Test evaluation program

Enter the name of the test file: _______________
```

If you enter the name of a valid executable test file, EVALUATE will continue. If the file does not exist or is not a valid executable test file, EVALUATE will respond with an appropriate error message and will again ask you to name the test file. You can terminate EVALUATE at this point by entering STOP.

EVALUATE will then ask for an output file:

```
     Enter the name of the file on which to write the report: ____________
```

If the file already exists, EVALUATE will ask you whether to erase it:

```
     File XXXXXXXX.XXX already exists -- Can it be erased (Y/N)? __
```

If you respond with a "Y," EVALUATE will erase the file and proceed. If you type "N," EVALUATE will ask you for another file name.

When you have entered the file name, EVALUATE will ask you for the mean and standard deviation of ability of the group to which the test will be administered.

```
     Enter the mean ability level of the group to be tested: _____

     Enter the standard deviation: _____
```

The group on which the items are calibrated is assumed to have a mean of 0.0 and a standard deviation of 1.0, so the new group should be described relative to the calibration group. EVALUATE assumes that the distribution of ability is normal with these parameters.

A long test with many subtests can produce a large volume of output. EVALUATE therefore asks whether you want all of the output. The high-volume components of the output are the test and subtest information functions. EVALUATE will ask if you want them.

```
     Would you like the test information function? (Y/N) : __

     Would you like the subtest information functions (Y/N) : __
```

After you answer these two questions, EVALUATE will analyze the test file. The output is written to the file you specified.

7.3. Interpreting the Output

A sample output file for a simple test consisting of two conventional subtests is shown in Figure 7-1. Only the test information function was requested.

The output contains reports on the two subtests individually and combined as a single test. In this example, the first subtest (OLDTEST) contained 15 items, had an estimated reliability of .677 in a population with ability distributed normally with the specified mean of 0.5 and standard deviation of 1.0. The _a_, _b_, and _c_ parameter means are as shown. The total test information is the total area under the test information curve. The expected information is the average of the test information function

Figure 7-1. Sample EVALUATE Report

```
Evaluate  -- Test evaluation for test file DEMTEST.EXC

   POPULATION CHARACTERISTICS

   Mean: 0.500
   Standard deviation : 1.000

   SUBTEST OLDTEST

   Number of items:  15
   Estimated reliability :  .677
   Average a parameter:  1.222
   Average b parameter:   .131
   Average c parameter:   .178
   Total test information :  XX.XXX
   Expected test information :  XX.XXX

   SUBTEST NEWTEST

   Number of items:  20
   Estimated reliability :  .774
   Average a parameter:  1.311
   Average b parameter:   .126
   Average c parameter:   .185
   Total test information :  XX.XXX
   Expected test information :  XX.XXX

   TOTAL TEST

   Number of items:  35
   Estimated reliability :  .XXX
   Average a parameter:  1.273
   Average b parameter:   .128
   Average c parameter:   .182
   Total test information :  XX.XXX
   Expected test information :  XX.XXX
```

Figure 7-1 (Continued)

TOTAL TEST INFORMATION FUNCTION

```
      I
      I
      I
      I                                    ...
I     I                                  ..   .
N     I                                ..      ..
F     I                               .          ..
O     I                            ...             .
R     I                          ..                 ..
M     I                     .....                     .
A     I                .....                           ..
T     I            ....                                  ....
I     I           .                                          .
O     I          .                                            .
N     I        ..                                              .
      I     ...                                                 .
      I    .                                                     ..
      I....
      I
       -------------------------------------------------------------
    -3.0      -2.0      -1.0       0.0       1.0       2.0       3.0

                                  THETA
```

weighted by a normal density function having a mean of 0.500 and a standard deviation of 1.0.

The reports on the second subtest and on the total test can be interpreted similarly.

The total test information shows test information as a function of ability between values of -3.0 and 3.0. This function, the sum of the item information functions in the individual subtests, can only be interpreted when all of the subtests measure the same trait.

7.4 Summary

The system allows you to predict the test characteristics througn IRT using the EVALUATE program. EVALUATE prompts you to enter necessary information and provides statistical pre-evaluation of the test specified.

As you recall from the introduction of this chapter, items must be calibrated before a test can be pre-evaluated. Calibration was covered in Chapter 6.

CHAPTER 8. DESIGNING NEW TESTING STRATEGIES

8.1. Introduction

As you will recall from earlier chapters, a testing strategy is a particular collection of test items sequenced in a particular way. So far, this manual has covered only the pre-defined and partially defined strategies provided with the system. While some users will find these sufficient for their work, you may want to develop new strategies beyond those provided.

You may want to be able to specify branching procedures, scoring methods, and new templates. The facility provided for doing this is a language called the Computerized Adaptive Testing Language (CATL). It is the language used to specify the entire system, including the pre-defined templates discussed in Chapter 5.

CATL resembles other higher level programming languages in its declarative and conditional statements. It also provides statements designed especially for adaptive testing. This mixture provides the flexibility needed to design strategies tailored to specific testing needs.

The following statements in CATL are described in this chapter. They are listed in the order in which they are presented.

Module delimiters	TEST and ENDTEST
Basic statements	#, !, and *
Variables	local, global, and SET
Declarative statements	SETSCORE and TERMINATE
Output control statements	KEEP AND AUTOKEEP
Conditional statements	IF, ELSEIF, and ENDIF
Adaptive statements	SEARCH, ENDSEARCH, SEQUENCE, ENDSEQUENCE, and $
Menu statements	INSTRUCT, _, and _#

8.2. Module Delimiters

Each test or subtest you create must be bracketed by a TEST-ENDTEST pair. TEST precedes a name that identifies the test. This name can be of any length but only the first eight characters will be used. While each TEST must have a name, a name following ENDTEST is optional.

Tests can be nested, one within the other, to a depth of ten. To ensure that each TEST has a corresponding ENDTEST, it is a good idea to include the optional ENDTEST name to allow the test consolidator to detect unmatched TEST-ENDTEST pairs. Figure 7-1 shows a sample TEST specification. There is a major TEST, SAMPLE1, which includes all of the subtests. On the next level, there are three subtests, SUB1, SUB2, and SUB3. On the third level, under SUB3, there is test SUB3.1.

Figure 8-1. Example of Test Nesting

```
TEST SAMPLE1
    o
    o
    o
   TEST SUB1
        o
        o
        o
   ENDTEST SUB1
   TEST SUB2
        o
        o
        o
   ENDTEST SUB2
   TEST SUB3
        TEST SUB3.1
             o
             o
             o
        ENDTEST SUB3.1
        o
        o
        o
   ENDTEST SUB3
     o
     o
     o
ENDTEST SAMPLE1
```

CATL entries are made in free-format fields. Statements can appear anywhere on the line. If a statement is longer than one line, the first line must end with an ampersand (&). Although both indentation and ENDTEST names are optional, they make program verification and maintenance easier, as demonstrated in Figure 8-2.

Figure 8-2. Test Without Indentation or ENDTEST Names

```
TEST SAMPLE2
     o
     o
     o
TEST SUB1
     o
     o
     o
ENDTEST
TEST SUB2
     o
     o
     o
ENDTEST
TEST SUB3
     o
TEST SUB3.1
     o
     o
     o
ENDTEST
     o
ENDTEST
ENDTEST
```

The tests in Figure 8-2 are the same as those in Figure 8-1 but are much more difficult to read. It is unclear in Figure 8-2, for instance, which ENDTEST corresponds to each TEST statement.

8.3. Basic Statements

As in any programming language, CATL has a few important basic statements. The most important of these is #, which precedes item numbers and directs the test administration program (ADMIN) to administer the specified items. This is the most fundamental use of #. Figure 8-3 displays the simplest type of test.

Figure 8-3. Basic Test

```
TEST BASIC
   #ITM001
   #ITM002
   #ITM003
ENDTEST BASIC
```

When executed, TEST BASIC will administer items ITM001, ITM002, and ITM003 from the item bank.

Another important basic statement is the ! operator. It signals the compiler to ignore the remainder of the line, allowing the user to insert comments.

Comments may seem unimportant, since a test stripped of all comments will execute exactly as before. They are important, however, because they make programs easier to read and help in test verification and maintenance.

It is wise to use comments to internally document the version, use, and creation date of the program. Any subsequent changes should also be documented. Figure 8-4 shows a simple commented test.

Figure 8-4. Simple Test with Comments

```
TEST COMMENTS
!      This is a sample test, created to display
!      comments in a test, 82/04/16, T. Smythe.
   #ITM004
   #ITM005
   #ITM006 ! Item 006 added in 82/09/12, R. Carothers
ENDTEST COMMENTS
```

Another basic facility is *. It is used to include a file, typically a pool of items that have been refined as a group, for use in tests. An example of * follows in Figure 8-5.

Figure 8-5. Test Including a Pool of Items

```
TEST INCLUDE
!      This test will execute a pool of items as well
!      as the other individual items specified.
   #ITM007
   #ITM008
   *POOL402
ENDTEST INCLUDE
```

The test above will consist of items ITM007 and ITM008 as well as all of the items listed on the file named POOL402.

8.4. Variables

CATL provides both local and global variables and a SET statement for setting them. Local variables are used only within the TEST block in which they are declared. The "scope" of a local variable is that test, and any attempt to reference it outside of that test will fail. A local variable is NOT passed to subtests.

Global variables can be declared anywhere in the test and can be referenced anywhere. Global variable names begin with an @.

All CATL variables are initialized to zero (default), but you can assign any value to any variable using SET. More than one variable can be SET at a time if you separate the declarations with a comma. In Figure 8-6, the variables MEAN and VAR are used.

Figure 8-6. Examples of Variable Use

```
TEST VARIABLE
   SET @MEAN = 1.0
   !     Set global variable MEAN to 1.0
       #ITM009
       #ITM010
       TEST SUBTEST
          SET LMEAN = @MEAN, VAR = 3.0
          !     Local LMEAN set to global @MEAN whose value is 1.0.
          !     Notice that two variables are set here.
              #ITM011
              *POOL002
       ENDTEST SUBTEST
 ENDTEST VARIABLE
```

8.5. Declarative Statements

Three declarative statements are included in CATL: SETSCORE, TERMINATE, and AUTOKEEP. They are called declarative to distinguish them from the executable statements (which include all the remaining statements). "Declarative" means that they are not actually executed by the test administrator. They set up characteristics of a test or subtest that affect the way it is administered.

Scoring mechanisms are provided with the system. They calculate scores such as the time it takes the examinee to respond, Bayesian ability estimates, and percentage correct. Appendix (SCORE) lists all available scoring mechanisms and the required variables for each. To use one of these scoring mechanisms you must declare SETSCORE, the name of the score and the names of variables (in the correct order) that you will use.

The format for SETSCORE is:

```
SETSCORE nameofscore (variable(s))
```

The scoring mechanisms listed in Appendix (SCORE) specify how many variables are needed.

For example, Bayesian scoring requires a mean value and a variance value. The following example test uses Bayesian scoring.

Figure 8-7. Bayesian Scoring Example

```
TEST BAYES
   SETSCORE BAYESM (MEAN,VAR)
   !     Bayesian scoring mechanism
   !     requires mean and variance
   !     variables
   SET MEAN = @PRIORMN, VAR = @PRIORVAR
   !     Note that @PRIORMN and @PRIORVAR are global variables
   !     set in another test outside TEST BAYES.
          #ITM012
          #ITM013
          #ITM014
          #ITM015
          #ITM016
          #ITM017
ENDTEST BAYES
```

TERMINATE is another declarative statement. It can be used once in a test and takes logic as its argument. When the logic is satisfied, TERMINATE causes a jump to the next ENDTEST, ending the execution of the current test.

The logical expression begins with an opening parenthesis. It can contain any of the defined variables in a logical expression using the operators:

<	less than
<=	less than or equal to
=	equal to
><	not equal to
>=	greater than or equal to
>	greater than
AND	logical and
OR	logical or
NOT	logical complement

+	arithmetic add
-	arithmetic subtract
*	arithmetic multiply
/	arithmetic divide
(	opening parenthesis
)	closing parenthesis

Figure 8-8 shows a simple example of the TERMINATE statement.

Figure 8-8. Terminate Example

```
TEST TERMEXM
   SETSCORE PCORR(PROPCORR), NADMIN(N)
   !    Specify the variables to be used in conjunction with
   !    scoring mechanisms PCORR and NADMIN.
   TERMINATE (PROPCORR < .3 OR PROPCORR > .8 OR N = 10)
   !    Execute the following items until one of the
   !    conditions above is met.
        #ITM016
        #ITM017

            o
            o
            o

        #ITM029
 ENDTEST TERMEXM
```

TERMINATE can be placed anywhere in the test, as Figure 8-9 shows. It functions exactly the same, however, regardless of where it is placed.

Figure 8-9. TERMINATE at the End

```
TEST TERMEND
   SETSCORE BAYES(MEAN,VAR), TIME (ET)
   !    Both Bayesian and timing scores will be used
      #ITM016
      #ITM017
      #ITM018
      *POOL004
   TERMINATE (VAR < 0.001 OR ET > 1200)
   !    TEST untilend will execute the items listed above,
   !    including the items in POOL004, until the
   !    variance value reaches 0.001 OR the elapsed time (ET)
   !    is greater than 1200 tenths of seconds.
ENDTEST TERMEND
```

The TERMINATE declarative will function in the same way here as it did in the previous figure.

The AUTOKEEP declarative is described in the next section along with its executable form, KEEP.

8.6. Output Control

There are two simple output facilities in CATL: the KEEP and the AUTOKEEP statements. KEEP writes the variables and strings of characters to the disc for storage. It is important to KEEP any data (e.g., scores) needed to analyze the test results.

Although writing scores and other data to a disc is adequate for storing the data, numbers in isolation can be difficult to interpret later. It is a good idea to precede each data KEEP with an identifying character string. More than one variable or string may be kept on the same line if they are separated by commas.

Figure 8-10 illustrates the use of KEEP in a moderately complex test.

Figure 8-10. KEEPing Score

```
TEST KEEPSCOR
!    This test consists of two subtests, one using
!    conventional items, one incorporating a pool
!    of items.  Scores from each are kept on the disc.
!
   TEST CONVENTL
      SETSCORE TIME(TENTHS)
      !   Assigning the variable TENTHS to the
      !   elapsed-time function.
          #ITM009
          #ITM010
          #ITM011
      KEEP "Length of time for three responses was:", SECONDS
      !   The character string will identify the
      !   numerical value in the data file.
   ENDTEST CONVENTL
!
   TEST POOLED
      SET MEAN = 1.0, VAR = 2.0
      SETSCORE BAYES(MEAN,VAR)
      !   Initialize local MEAN and VAR and specify
      !   Bayesian scoring with MEAN and VAR
          *POOL003
      KEEP "Bayesian MEAN score for POOL003 was:", MEAN
   ENDTEST POOLED
ENDTEST KEEPSCOR
```

AUTOKEEP is similar in format and function to KEEP. It is different in two respects: (1) it is automatically executed every time an item is administered, and (2) it is a declarative statement, which means that it is either on or off for the entire TEST block in which it is used. AUTOKEEP is useful whenever item response information must be kept. It is particularly useful in keeping data for item calibration. Figure 8-11 shows AUTOKEEP used in the template CALIBCON, which is used to set up a conventional test containing trial items for pre-calibration administration.

Figure 8-11. Template CALIBCON

```
!
! Special template for item calibration data collection
!
TEST CALIB
INSTRUCT Calibration template -- Use to collect data for item calibration.
INSTRUCT
!
! Set up scores to compute
!
SETSCORE NUMBER(ITNUM), KEY(ITKEY), RESPONSE(ITRESP)
!
! Use AUTOKEEP to automatically keep scores after each item administration.
!
AUTOKEEP ITNUM, ITKEY, ITRESP
!
! Read all item numbers to be included
!
INSTRUCT Enter the item numbers of the items to be included, one at a
INSTRUCT time and finish by pressing the return key alone.
# #
ENDTEST
```

8.7. Conditional Statements

IF is a conditional statement that tests for the truth of the subsequent argument. It takes the form:

    IF (logic)

Its utility is enhanced by the ELSEIF statement, which is also conditional and also requires logic for argument. An ELSEIF with no logic will be executed if none of the IF or ELSEIF logic above it is true. ENDIF simply indicates the end of the block of conditional execution.

Figure 8-12 shows an example of a simple use of IF, ELSEIF, and ENDIF.

Figure 8-12 -- Simple IF, ELSEIF, and ENDIF

```
IF (MEAN < -1.0)
  !IF MEAN SCORE IS LESS THAN -1.0, ADMIN ITM001
    #ITM001
ELSEIF (MEAN > -1.0)
  !IF MEAN SCORE IS GREATER THAN -1.0, ADMIN ITM002
    #ITM002
ELSEIF
  !IF MEAN SCORE IS EQUAL TO 1.0, ADM ITM003
    #ITM003
ENDIF
```

8.8. Adaptive Statements

Adaptive statements are specific to adaptive testing strategies. They direct the system to use a variable or constant to find the most informative items, to differentially branch after correct or incorrect responses, and to branch to other parts of the test specification. This section also covers the # symbol in more detail.

The SEARCH statement searches a pool of test items and determines which unadministered item has the most psychometric information at an ability equal to a value specified. The SEARCH statement includes a variable or constant specifying the search value. The item pool is delimited by the SEARCH statement and an ENDSEARCH statement. An example of a Bayesian test is shown in Figure 8-13.

As discussed in Section 8.3., the # character followed by an item number administers an item when the statement is executed. The item statement may be followed by any of the following characteristics. If all the characteristics you choose to use do not fit on one line, an & can be used to continue the line.

CORRECT: item or label	If the item is answered correctly, execution continues at the item or label specified. This is known as branching. The item or label specified must be within the test currently executing. The label referenced must begin with a $.
INCORRECT: item or label	If the item is answered incorrectly, execution branches to the item or label specified.

Figure 8-13. A Searched Bayesian Test

```
TEST BAYES
!
!   Specify the Bayesian score with a mean of "MEAN"
!   and a variance of "VAR."
!
   SETSCORE BAYES(MEAN,VAR)
!
!   Set the Bayesian prior.
!
   SET MEAN = 0.0, VAR = 1.0
!
!   Search through the items in pool POOL102
!
   SEARCH MEAN
   *POOL102
   ENDSEARCH
!
!   Quit when the posterior variance is less than 0.01.
!
   TERMINATE (VAR < .01)
!
!   Keep the final Bayesian scores.
!
   KEEP MEAN,VAR
ENDTEST BAYES
```

ALT: item or label, item or label, ...

If the first alternative is selected when the test is administered, it branches to the first label; if the second alternative is selected, execution branches to the second label; and so on.

Some adaptive strategies, such as the stradaptive strategy, require several strata or sequences of items. Each time a stratum is branched to, the next unadministered item in sequence is presented. The SEQUENCE statement, along with the ENDSEQUENCE statement, allows a sequence of items to be specified.

A SEQUENCE statement can be used much like an item for branching. When it is branched to, it administers an item and then follows the specified logic to branch to another item or label outside the sequence. The sequence is terminated with an ENDSEQUENCE. Figure 8-14 gives an example of a two-stratum stradaptive test implemented using sequences.

Figure 8-14. Use of the Sequence Statement

```
TEST STRAT
!
!        Select Bayesian scoring.
!
   SETSCORE BAYES(MEAN,VAR)
!
!        Set Bayesian prior.
!
   SET MEAN = 0.0, VAR = 1.0
!
!        Plan to Quit when variance is less than 0.01.
!
   TERMINATE (VAR < 0.01)
!
!        First sequence will execute first because
!        of its location, and subsequently after every
!        incorrect response.
!
     $SEQ1 SEQUENCE (IN:$SEQ1, CO:$SEQ2)
!
!        This branch logic is only for the SEQUENCE
!
          #ITM001
          #ITM002
             o
             o         !  Branch logic will be ignored
             o         !  within the sequence itself.
          #ITM010
     ENDSEQUENCE
!
!        Second SEQUENCE will execute after every
!        correct response.
!
     $SEQ2 SEQUENCE (IN:$SEQ1, CO:$SEQ2)
          #ITM011
          #ITM012
             o
             o
             o
          #ITM020
     ENDSEQUENCE
!
!        Keep MEAN and VAR.
!
     KEEP MEAN, VAR
ENDTEST STRAT
```

8.9. Menu Statements

Now all the tools necessary to write CATL programs that will execute tests have been introduced. With a few more menu statements, CATL templates (such as the pre-defined ones described in Chapter 5) can be written. Several versions of the same test can be easily created with these templates, without duplicating the CATL code each time. The only additional statements necessary are INSTRUCT, _, and _#.

INSTRUCT is followed by a character string you wish to output to the template user. These are generally instructions for filling in items and scores.

The _ symbol is used to mark where these responses will be entered. If more than one response line may be entered by the template user, _# is used. This will accept multiple responses and put each on a separate line. The multiple entry is terminated by a carriage return alone. Figure 8-15 shows a typical template.

8.10 Summary

CATL, like any other programming language, requires rigor in use. However, unlike most languages you may have experience using, CATL has been specifically designed for use in computerized adaptive testing. CATL frees the user from the time consuming lower level details of implementation.

Before a test can be administered, it must be consolidated. Consolidation was discussed in Chapter 5.

Figure 8-15. Fixed Length Conventional Template

```
!       Fixed Length Conventional Test Template
!
!       This instruct is the first thing the template
!       user gets when template "vartempl" executes.
!
INSTRUCT Conventional test template -- Enter test title.
!
!       The test title is entered.
!
TEST _
!
!       The template sets up the scores to keep.
!
SETSCORE PCORR(PROPORTION), FORMSCORE(FORMULA)
!
!       Now the template requests test item numbers.
!
INSTRUCT Enter test item numbers, one per line.  When
INSTRUCT you have entered them all, press the carriage
INSTRUCT return alone.
#_#
!
!       The # symbol indicates that more than one item
!       number may be entered.
!
INSTRUCT Scores computed are PROPORTION (proportion correct)
INSTRUCT and FORMULA (formula score).
INSTRUCT List the scores you would like to keep, separated
INSTRUCT by commas.
KEEP _
!
!       This KEEP will put the scores the user wants to keep
!       out to the disc.
!
ENDTEST
!       This ends the test specified by the user.
```

CHAPTER 9. DISPLAYING TEST RESULTS

9.1. Introduction

After a test has been administered you will undoubtedly want to see the results. Previous chapters have discussed how to develop a test, how to administer it, and how to keep test scores on a file. The system also contains a facility for translating the raw numbers into any of several forms. Two programs, DISPGEN and DISPEX, constitute this facility.

DISPGEN takes an interpretive language that you provide and translates it into a form that the system can readily process. DISPEX uses the translated form and an examinee's data file and displays the test results according to your specifications. Four steps are required to produce a display: (1) the test specification must be set up to keep the raw data in an appropriate format, (2) a display specification must be created in a manner similar to creation of a test specification, (3) the display specification must be translated by the DISPGEN program, and (4) the display must be produced by the DISPEX program.

9.2. Keeping the Required Data

The test executor keeps data on a file as requested by the test specification. Each KEEP statement writes one line to the file and each variable listed in the KEEP statement occupies eight spaces on the line. A label will occupy as many eight-space fields as it needs. For example, an eight-character label requires one field or eight spaces; a nine-character label requires two fields or sixteen spaces.

The scores to be displayed must be read by the display program from the file produced by the test administrator. The scores must be recorded unambiguously on that file. This is easy if only one line of scores is kept. If several lines are kept, recording is still easy if the number of lines is constant and the same set of scores is kept for all examinees. In either of these cases, the sequence and location of scores is unambiguous. However, if the number and/or sequence of scores can vary, the scores must be identified.

The test specification shown in Figure 9-1 will always produce two lines on the KEEP file and all of the scores will be in a known location.

When an examinee is tested, the test administrator will create a data file that includes two lines of data. The proportion correct will always be in the first eight spaces of the first line, and the Bayesian mean and variance will be in the first sixteen spaces of the second line.

When a variable number of lines are to be written, it is difficult to identify the variables by position. The AUTOKEEP statement used with a

Figure 9-1. A Constant Data File

```
!
! Conventional test with Bayesian scoring
!
TEST CONBAYES
!
! Compute conventional and Bayesian scores.
!
SETSCORE PCORR(PROPORTION), BAYES(MEAR,VAR)
#ITM001
#ITM002
#ITM003
#ITM004
#ITM005
!
! KEEP PROPORTION CORRECT ON FIRST LINE
!
KEEP PROPORTION
!
! KEEP BAYESIAN MEAN AND VARIANCE ON SECOND LINE
!
KEEP MEAN, VAR
ENDTEST
```

variable-length adaptive test will write an unpredictable number of lines and the only way to separate the variables is to label the lines. The test specification in Figure 9-2 labels the lines in the KEEP file and allows the scores to be located.

On the KEEP file for this test, all lines with "RESPONSE" in the first field will contain item responses, the line with "PCORR" in the first field will contain the proportion-correct score, and the line with "MEANVAR" in the first field will contain the two Bayesian scores.

9.3. Creating the Display Specification

The display specification consists of two elements: score correspondence and interpretation. The first element, score correspondence, tells the display program where to find the scores and what to call them. The second produces interpretive comments and/or values resulting from the scores.

The score correspondence section begins with the first line of the specification and continues until the beginning of the interpretation section. There are two ways in which correspondences can be made.

First, scores can be listed on a line-by-line basis without labels with a one-to-one correspondence between lines in the file and lines in the

Figure 9-2. A Labeled Data File

```
!
! Bayesian test
!
TEST CONBAYES
!
! Compute conventional and Bayesian scores.
!
SETSCORE PCORR(PROPORTION), BAYES(MEAR,VAR), RESPONSE(ITRESP), &
NUMBER(ITNUM)
!
! Initialize Bayesian scores
!
SET MEAN=0.0, VAR=1.0
!
! Terminate if variance is less than 0.1
!
TERMINATE (VAR < 0.1)
!
! Keep item responses after each item is administered
!
AUTOKEEP "RESPONSE",ITNUM,ITRESP
!
! SEARCH ON THE BAYESIAN MEAN
!
SEARCH MEAN
#ITM001
#ITM002
#ITM003
#ITM004
#ITM005
#ITM006
#ITM007
#ITM008
#ITM009
ENDSEARCH
!
! KEEP PROPORTION CORRECT ON FIRST LINE
!
KEEP "PCORR",PROPORTION
!
! KEEP BAYESIAN MEAN AND VARIANCE ON SECOND LINE
!
KEEP "MEANVAR",MEAN, VAR
ENDTEST
```

correspondence section of the specification. For example, the following correspondence would work well with the conventional test specified in Figure 9-1:

```
PROPORTION
MEAN, VAR
```

No labels are required. The display program expects to find a value to assign to PROPORTION in the first field of the first line and values for MEAN and VAR in the first two fields of the second line.

Second, if scores are to be located by a label, as in the example shown in Figure 9-2, the appropriate label and score names are included. For example,

```
"MEANVAR", MEAN, VAR
```

would search for a line in the Keep file beginning with the label "MEANVAR" and would assign the values found in the two fields following the label to the variables MEAN and VAR, respectively. The Keep file should contain only one line with the specified label. If several lines are so labeled, unpredictable results may occur. Labeled and unlabeled correspondences should not be mixed in the specification because the results are unpredictable.

The second section of the specification describes how to display the scores. It is built by concatenating modules of display instructions. A module consists of the following structure:

(1) An opening bracket	[
(2) A module number (optional)	MOD001
(3) Logic (optional)	^(logical expression)^
(4) Display statements	
A. Text	The client is very bright.
B. Module calls	#MOD002
C. Scores to print	?MEAN
D. Header lines	@Title@
(5) Other modules	[------]
(6) A closing bracket	]

The module number is a combination of up to eight letters and up to three digits, like an item number. It is optional and must follow immediately after the opening bracket of the module. It is neccessary only if the module is to be called and, in that case, serves as a module identifier.

The logical expression begins with a caret and an opening parenthesis. It can contain any of the named scores in a logical expression using the operators:

<	less than
<=	less than or equal to
=	equal to
><	not equal to
>=	greater than or equal to
>	greater than
AND	logical and
OR	logical or
NOT	logical complement
+	arithmetic add
-	arithmetic subtract
*	arithmetic multiply
/	arithmetic divide
(	opening parenthesis
)	closing parenthesis

If the logical expression is absent or is evaluated and found to be true, the module is executed. Otherwise, the module is skipped.

The display statements can consist of a combination of text and three other operators. The text is printed. If a # is found, the program expects a module number to follow and that module is inserted in its entirety. If a question mark is found, a score name is expected to follow and it is printed in the text, occupying eight spaces with a decimal and three digits to the right of the decimal. An @ forces the beginning of a new line of text.

Other modules may be embedded within a module. They are executed as they are encountered. An embedded module is treated the same as a called module.

The display program will sequentially execute only the first module on the display specification. This means that additional modules can be included at the end of the specification and called when they are needed. They will not be executed unless they are called.

Figure 9-3 shows a sample display specification for the conventional test with Bayesian scoring discussed above.

Another display specification for the same data is shown in Figure 9-4.

In the first example, the display only lists the scores. In the second, interpretive statements are made regarding the scores.

Figure 9-3. Display Specification for a Conventional Test

```
PROPORTION
MEAN, VAR
[
The proportion correct score achieved was ?PROPORTION @
  @
  @
  @
The Bayesian mean was ?MEAN and the Bayesian posterior variance
was ?VAR @
]
```

Figure 9-4. Alternate Specification for a Conventional Test

```
PROPORTION
MEAN, VAR
[
   [^(MEAN < -1.0)^
   The examinee did not do well on the test.
   ]
   [^(MEAN >= -1.0 AND MEAN <= 1.0)^
   The examinee performed at an average level.
   ]
   [^(MEAN > 1.0)^
   The examinee did well on the test.
   ]
   [His/her Bayesian score was ?MEAN .
   ]
]
```

9.4. Translating the Specification

The display specification is translated by the program DISPGEN. To run DISPGEN, enter the command:

```
DISPGEN
```

The system will respond with:

```
Display translator -- Version X.X

Enter the specification file name: _______________
```

You should respond by entering the name of the file that contains the display specification. The system will then respond with:

```
Enter the output file name: _______________
```

You should enter the file on which to write the translated output. DISPGEN will then translate the input file and write it to the output file. If any errors in the logic or the text are discovered, the line with the error and a pointer to the location will be printed on the terminal in the following format:

```
     Error on line 137 of input near pointer:

     [^(SCORE > 50) #MOD002]
                    ^
```

In this example, the closing caret was omitted from the logic line and the error was detected when the # character was read.

If no errors are encountered, the following message will be displayed on the terminal:

```
     XXX lines of text processed
```

9.5. Executing the Display

The display is executed by running the program DISPEX, which is run by entering the command:

```
     DISPEX
```

The system will respond with:

```
     Display executor -- Version X.X

     Enter the name of the translator output file: _______________
```

You should respond by entering the name of the output file produced by DISPGEN. Next, the system will ask where the display should be printed:

```
     Display to terminal (T) or printer (P) ? __
```

You should respond with a T or a P. The system will then respond by asking for a data file:

```
     Enter the name of the examinee data file: _______________
```

When you enter the file name, the display will be produced at the specified terminal. Upon completion, it will ask for another examinee's data file. You can terminate DISPEX by entering STOP.

APPENDIX B. PARTIAL BNF DESCRIPTION FOR CATL

BNF is a formal system for specifying the syntax of a programming language. A complete BNF description of a language describes what statements are legal in the language and how to parse the language using a compiler. The description provided below is not complete: It is complete from the top of the language hierarchy to the level of the statement; it does not define all of the clauses or elements within each of the statements. Furthermore, only the executable statements are described; template statements and compiler utilities (e.g., the include statement) are excluded from this description.

In the description, the following notation is used:

<construct>	A symbolic language element.
::=	A replacement operator interpreted to mean that the construct on the left becomes the construct on the right.
|	A vertical line separating alternative constructs
[construct]	An optional language element.
( )	Grouping operators.

```
<a test>                  ::= TEST <name> [<declarative statements>]
                                [<statements>] ENDTEST

<declarative statements>  ::= <declarative statement>
                                [<declarative statements>]

<declarative statement>   ::= <TERMINATE statement> |
                              <SETSCORE statement> |
                              <AUTOKEEP statement>

<TERMINATE statement>     ::= TERMINATE <logical expression>

<SETSCORE statement>      ::= SETSCORE <scorename> <variable list>

<AUTOKEEP statement>      ::= AUTOKEEP <variable list>

<statements>              ::= <statement> [<statements>]
```

```
<statement>               ::= <IF statement> |
                              <item statement> |
                              <KEEP statement> |
                              <SEARCH statement> |
                              <SEQUENCE statement> |
                              <SET statement>

<IF statement>            ::= IF <logical expression> <statements>
                                [<elses>] ENDIF

<elses>                   ::= <ELSEIF clause> [<elses>]

<ELSEIF clause>           ::= ELSEIF <logical expression> <statements>

<item list>               ::= <item statement> [<item list>]

<item statement>          ::= # <item identifier> [<branches>]
                                [<characteristics>]

<KEEP statement>          ::= KEEP <variable list>

<SEARCH statement>        ::= SEARCH <variable or constant> <item list>
                                ENDSEARCH

<SEQUENCE statement>      ::= SEQUENCE [<branches>] [<item list>]
                                ENDSEQUENCE

<SET statement>           ::= SET <variable> = <expression>
```

APPENDIX C. LIST OF SYSTEMS CONSIDERED

The following systems were too expensive. This was due, in part, to the fact that they were sixteen-bit microprocessors. Sixteen-bit microprocessors are much more powerful than the eight-bit processors, however they are still very new and fairly expensive. Sixteen-bit microprocessor-based systems usually contain more memory than eight-bit systems which also increases the cost. These systems would be logical choices for testing applications in which high speed is more important than low cost or where multi-user systems are desired.

System 8305
Dual Systems
720 Channing Way, Berkeley, CA 94710
Processor: 68000 microprocessor
Memory: 256K RAM
Discs: Two dual-density single-sided eight inch floppies
Bus: S-100
Operating System: UNIX
Price: $8,295

AMPEX Dialogue 80
Computex
5710 Drexel Avenue, Chicago, IL 60637
Processor: Z8000 microprocessor
Memory: 256K RAM
Discs: Two eight inch floppies
Bus: Multibus compatible
Operating System: Custom multiuser
Price: $7,053

CD100
Callan Data Systems
2637 Townsgate Rd., Westlake, CA 91361
Processor: LSI-11 or Z80 microprocessor
Memory: 64K RAM
Discs: Two 5 1/4 inch floppies
Bus: Multibus or Q-bus
Operating System: RT-11, CP/M
Price: $7,465

TEC-86
Tecmar, Inc.
23600 Mercantile Rd., Cleveland, OH 44122
Processor: 8086 microprocessor
Memory: 64K RAM
Discs: Two eight inch floppies
Bus: S-100
Operating System: CP/M 86
Price: $4,390; no terminals

System 2
Seattle Computer
1114 Industry Dr., Seattle, WA 98188
Processor: 8086 microprocessor
Memory: 128K RAM
Discs: dual density floppy controller; no drive
Bus: S-100
Operating System: MS-DOS
Price: $4,785.50; no terminal

DS990
Texas Instruments
P.O. Box 202129, Dallas, TX 75220
Processor: 9900 microprocessor
Memory: N/A
Discs: Two eight inch floppies (1.2M)
Bus: N/A
Operating System: UCSD Pascal
Price:

HP1000 L-Series Model 5
Hewlett-Packard
11000 Wolfe Rd., Cupertino, CA 95014
Processor: L-series processor
Memory: 64K RAM
Discs: 450K bytes of floppy
Bus: L-series
Operating System: RTE-L, RTE-XL
Price: $8,100

ERG68-696
Empirical Research Group
P.O. Box 1176, Milton, WA 98354
Processor: 68000 microprocessor
Memory: 64K RAM
Discs: two eight inch dual-density floppies
Bus: S-100 (IEEE)
Operating System: FORTH, IDRIS
Price: $7,395

Model 235
Zendex
6644 Sierra Lane, Doublin, CA 94566
Processor: 8088 or 8085 microprocessor
Memory: 64K RAM
Discs: two floppies
Bus: Multibus
Operating System: MP/M II
Price: $7,595; no terminals

ACS8000
Altos Computer Systems
2360 Bering Dr., San Jose, CA 95131
Processor: Z80 or 8086 microprocessor
Memory: 64K RAM
Discs: Two eight inch floppies (1 M)
Bus: Multibus
Operating System: CP/M, OASIS, MPMII, CP/M-86, MP/M-86, OASIS-16, XENIX
Price: $3,650; no terminals

MP/100
Data General
Westboro, Mass. 01581
Processor: mN602 microprocessor
Memory: 64K RAM
Discs: Two 5 1/4 inch floppies (630K)
Bus: N/A
Operating System: MP/OS, DOS, RTOS
Price: $9,490

SuperSixteen
Compupro/Godbout Electronics
Oakland Airport, CA 94614
Processor: 8088 and 8085 microprocessor
Memory: 128K RAM
Discs: no drives
Bus: S-100 (IEEE)
Operating System: CP/M, CP/M-86
Price: $3,495; no terminals

VICTOR-9000
Victor Business Products
3900 North Rockwell St, Chicago, IL 60618
Processor: 8088 microprocessor
Memory: 128K RAM
Discs: Two eight inch single-sided floppies (1.2M)
Bus: N/A
Operating System: CP/M-86, MS-DOS
Price: $4,200

The following systems have sixteen-bit microprocessors and special features not available in most systems. Although these special features make the system too expensive, more than $3,000, these systems would be excellent choices where their special features were required.

CGC-7900
Chromatics
2558 Mountain Industrial Blvd., Tucker, GA 30084
Processor: 68000 or Z80 microprocessor
Memory: 128K RAM
Discs: Two eight inch floppy (1M)
Bus: N/A
Operating System: IDRIS, DOS, CP/M
Price: $19,995

DISCOVERY
Action Computer Enterprises
55 West Del Mar Blvd., Pasadena, CA 91105
Processor: 8086 and Z80 microprocessors
Memory: 192K RAM
Discs: Two eight inch floppies
Bus: S-100
Operating System: DPC/OS, CP/M
Price: $6,000 + $1,395 per user

The following systems closely resemble the systems in the "acceptable" category. In most cases, the only reason these systems were excluded from further consideration was their high cost.

EAGLE II
AVL
503A Vandell Way, Campbell, CA 95008
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: Two 5 1/4 inch dual-sided double-density (1M)
Bus: N/A
Operating System: CP/M
Price: $3,995

M System
Business Operating Systems
2835 East Platte Ave., Colorado Springs, CO 80909
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: two eight inch double-density floppies
Bus: S-100
Operating System: CP/M, MP/M, CPNET, BOS
Price: $5,000

Model 500
Columbia Data Products
8990 Route 108, Columbia, MD 21045
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: two 5 1/4 inch single-sided double-density drives (320K)
Bus: N/A
Operating System: CP/M, FDOS
Price: $4,495

Z2D
Cromemco
280 Bernardo Ave., Mountain View, CA 94040
Processor: Z80A microprocessor
Memory: 64K RAM
Discs: Two 5 1/4 inch floppies (780K)
Bus: S-100
Operating System: CROMIX, CDOS
Price: $3,990

Term Master add on
Barreto & Associates
P.O. Box 874, Sedalia, Missouri 65301
Processor: Z80 microprocessor
Memory: 32K RAM
Discs: One 5 1/4 inch floppy (300K)
Bus: N/A
Operating System: CP/M, MP/M
Price: $3,850

System 80
Exidy Systems
1234 Elko Dr., Sunnyvale, CA 94086
Processor: Z80 microprocessor
Memory: 48K RAM
Discs: two quad-density 5 1/4 inch floppies (616K)
Bus: S-100
Operating System: CP/M
Price: $4,490

T-200
Toshiba America
2441 Michelle Dr., Tustin, CA 92680
Processor: 8085A microprocessor
Memory: 64K RAM
Discs: Two 5 1/4 inch floppies (560K)
Bus: N/A
Operating System: CP/M, BASIC
Price: $4,500

VIP
Vector Graphics
500 N. Ventu Park Rd., Thousand Oaks, CA 91320
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: single-sided, quad density 5 1/4 floppy (315K)
Bus: S-100
Operating System: CP/M
Price: $3,995

DB8/4
Dynabyte
1005 Elwell Ct., Palo Alto, CA 94303
Processor: Z80 microprocessor
Memory: 48K RAM
Discs: Two eight inch drives (2M)
Bus: S-100
Operating System: CP/M, MPM
Price: $4,495; no terminal

Model 525
Ithaca Intersystems
1650 Hanshow Rd., Ithaca, NY 14850
Processor: Z80 microprocessor
Memory: 128K RAM
Discs: two single-sided 5 1/4 inch floppies
Bus: S-100
Operating System: Custom, CP/M, MP/M, COHERENT
Price: $5,595

Advantage, Horizon
Northstar Computers
14440 Catalina St., San Leandro, CA 94577
Processor: Z80
Memory: 64K RAM
Discs: two double-density floppies
Bus: S-100
Operating System: CP/M
Price: $3,996 (Advantage); $3,830 (Horizon)

QDP-100
Quasar Data Products
10330 Brecksville Rd., Cleveland, OH 44141
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: two eight inch double-sided double-density floppies (2.4M)
Bus: S-100
Operating System: CP/M, CBASIC
Price: $4,695

System 2812
Systems Group
1601 Orangewood Ave., Orange, CA 92668
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: Two single-sided floppies
Bus: S-100
Operating System: CP/M, MPM, OASIS
Price: $5,035

SPRINT 68
Wintek
1801 South St., Lafayette, IN 47904
Processor: 6800 microprocessor
Memory: 48K RAM
Discs: Two eight inch floppies
Bus: N/A
Operating System: WIZRD
Price: $3,949

S-11
Zobex
P.O. Box 1847, San Diego, CA 92112
Processor: Z80A microprocessor
Memory: 64 K RAM
Discs: two eight inch floppies
Bus: S-100
Operating System: CP/M, MPM
Price: $4,900; no terminal

P300-01
California Computer Systems
250 Caribbean Dr., Sunnyvale, CA 94086
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: 1.2 megabytes of floppy
Bus: S-100
Operating System: CP/M, OASIS
Price: $5,695

56K B System
Gimix
1337 W. 37th Place, Chicago, IL 60609
Processor: 6809 microprocessor
Memory: 56K RAM
Discs: no disc drives included
Bus: SS-50
Operating System: OS-9, FLEX
Price: $2,988.59; no terminal

Model 5000
IMS International
2800 Lockhead Way, Carson City, NV 89701
Processor: Z80A microprocessor
Memory: 64K RAM
Discs: Two 5 1/4 inch dual-sided double-density floppies
Bus: S-100
Operating System: CP/M, MPM, TurboDOS
Price: $3,695

H-89
Heath
Dept 334-846, Benton Harbor, MI 49022
Processor: dual Z80 microprocessors
Memory: 64K RAM
Discs: three single-sided 5 1/4 inch drives (480K)
Bus: N/A
Operating System: CP/M, HDOS
Price: $4,095

YX3200
Sharp Electronics
10 Keyston Place, Paramus, NJ 07652
Processor: N/A
Memory: 64K RAM; 32K ROM
Discs: Two 5 1/4 inch floppies (570K)
Bus: N/A
Operating System: CP/M, FDOS
Price: $4,495

The following eight-bit systems have special features which might make them desirable for some applications. They do, however, cost more than $3,600.

8052/8053
Intelligent Systems
225 Technology Park, Norcross, GA 30092
Processor: 8080A microprocessor
Memory: 8K RAM
Discs: Two 5 1/4 inch (8052, 160K) or eight inch (8053, 590K) floppies
Bus: N/A
Operating System: CP/M
Price: $4,795 (8052); $5,870 (8053)

Millie
MicroDaSys
2811 Wilshire Blvd., Santa Monica, CA 90403
Processor: Z80A microprocessor
Memory: 64K RAM
Discs: two eight inch single-sided double-density floppies (1M)
Bus: S-100
Operating System: CP/M
Price: $9,995

Mariner
Micromation
1620 Montgomery St., San Francisco, CA 94111
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: Two double-sided double-density floppies (2M)
Bus: S-100
Operating System: CP/M, MP/M
Price: $5,500

Zeus
OSM Computer
2364 Walsh Ave., Santa Clara, CA 95051
Processor: Z80A microprocessor
Memory: 64K RAM
Discs: Two eight inch floppies (1.2M)
Bus: S-100
Operating System: MUSE (CP/M)
Price: $5,900 + $3,400 per user

The following systems can be configured with two floppy discs and a terminal (or built-in keyboard and screen) for less than $3,000. However, all of the following systems lack disc space, memory space, or a good software base.

CBM 8032
Commodore Computer Systems
681 Moore Rd., King of Prussia, PA 19406
Processor: 6508 microprocessor
Memory: 32K RAM
Discs: Two 5 1/4 inch drives (340K)
Bus: N/A
Operating System: Custom
Price: $3,095 (340K); $3,590 (1M)

Mod 5
Compucolor
P.O. Box 569, Norcross, Georgia 30071
Processor: 8084 microprocessor
Memory: 32K RAM
Discs: Two single-sided single-density floppies (102.4K)
Bus: N/A
Operating System: BASIC
Price: $2,490

C8PDF
Ohio Scientific
1333 S. Chillicothe Rd., Aurora, OH 44202
Processor: 6502A microprocessor
Memory: 32K RAM
Discs: dual eight inch floppies
Bus: Custom
Operating System: OS65U
Price: $3,495

*
Smoke Signal Broadcasting
31336 Via Colinas, Westlake Village, CA 91361
Processor: 6800 or 6809 microprocessor
Memory: 32K RAM
Discs: Two 5 1/4 inch single-sided, double-density floppies (320K)
Bus: SS-50
Operating System: DOS68, DOS69
Price: $3,095

TI-99/4
Texas Instruments
P.O. Box 202129, Dallas, TX 75220
Processor: 9900 microprocessor
Memory: 16K RAM
Discs: three 5 1/4 inch single-sided, single-density floppies (270K)
Bus: N/A
Operating System: UCSD Pascal, COBOL, LOGO
Price: $2,825

The following systems fulfill all the requirements for adaptive testing application. Furthermore, they offer a better cost per value ratio that most of the other systems evaluated. They are all produced by small companies and must be shipped back to the manufacturer for repair. Unfortunately, they do not have a standard bus and are thus less expandable than many other systems.

ACT-85 System
Autocontrol
11744 Westline Ind. Dr., St. Louis, MO 63141
Processor: 8085 microprocessor
Memory: 64K RAM
Discs: Two eight inch double-sided floppies
Bus: N/A
Operating System: CP/M
Price: $2,750

EXO Z80
Micro Business Associates
500 Second Street, San Francisco, CA 94107
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: Two eight inch floppies (1.2M)
Bus: N/A
Operating System: Custom
Price: $2,995

LNW 80
LNW Research
2620 Walnut St., Tustin, CA 92680
Processor: Z80 microprocessor
Memory: 48K RAM
Discs: one floppy
Bus: N/A
Operating System: CP/M, CDOS, TRS-DOS
Price: $1,915

MOD III
Microcomputer Technology
3304 W. Macarthur, Santa Ana, CA 92704
Processor: Z80 microprocessor
Memory: 48K RAM
Discs: dual double-sided double-density floppies (1.5M)
Bus: none
Operating System: CP/M, RS-DOS
Price: $2,799

Model 500T
Quay
P.O. Box 783, 527 Industrial Way West, Eatontown, NJ 07724
Processor: Z80A microprocessor
Memory: 64K RAM
Discs: Two double-density single-sided 5 1/4 inch floppies (400K)
Bus: N/A
Operating System: CP/M
Price: $2,995

The following systems have sufficient memory, sufficient disc space for most testing applications, an ample supply of existing software, a national maintenance service, and cost less than $3,600. They all are packaged as stand-alone systems with screens and keyboards included. They are thus all candidates for an adaptive testing system.

Apple II Plus
Apple Computer
10260 Brandley Dr., Cupertino, CA 95014
Processor: 6502 microprocessor
Memory: 48K RAM
Discs: Two 5 1/4 inch floppies (320K)
Bus: Apple
Operating System: DOS
Price: $2,700

IBM Personal Computer
IBM Information Systems Division
Entry Systems Business, P.O. Box 1328, Boca Raton, FL 33432
Processor: 8088 microprocessor
Memory: 64K RAM
Discs: Two 5 1/4 inch floppies (320K)
Bus: IBM
Operating System: MS/DOS, CP/M-86
Price: $3,290

SuperBrain
Intertec Data Systems
2300 Broad River Rd., Columbia, SC 29210
Processor: Z80 microprocessor
Memory: 64K RAM
Discs: Two 5 1/4 inch floppies (320K)/(700K)
Bus: S-100 Interface
Operating System: CP/M
Price: $2,895 (350K)/$3,595 (700K)

Model 8000
NEC America
1401 Estes Ave., Elk Grove Village, IL 60007
Processor: UPD780c-1 (Z80) microprocessor
Memory: 32K/64K RAM
Discs: dual 5 1/4 inch floppies (286.72 K)
Bus: N/A
Operating System: CP/M
Price: $2,700 (32K)/$3,300 (64K)

APC
NEC America
1401 Estes Ave., Elk Grove Village, IL 60007
Processor: 8086 microprocessor
Memory: 256K RAM
Discs: quad-density 8 inch floppy (1M)
Bus: N/A
Operating System: MS-DOS, CP/M-86
Price: $3,200

TRS-80 Model III
Radio Shack Division of Tandy
Fort Worth, TX 76102
Processor: Z80 microprocessor
Memory: 48K RAM
Discs: Two 5 1/4 inch double-density floppies (358K)
Bus: N/A
Operating System: BASIC, TRS/DOS
Price: $2,495

MBC-2000
Sanyo Business Systems
52 Joseph St., Moonachie, NJ 07074
Processor: dual 8085 microprocessors
Memory: 64K RAM
Discs: one or two 5 1/4 inch floppies (287K)/(574K)
Bus: Multibus interface
Operating System: CP/M, TS/DOS
Price: $1,995 (287K)/$3,495 (574K)

TS-802
Televideo Systems
1170 Morse Ave., Sunnyvale, CA 94086
Processor: Z80 miroprocessor (4 MHz)
Memory: 64K RAM
Discs: dual 5 1/4 inch floppies (1M)
Bus: N/A
Operating System: CP/M, MmmOST
Price: $3,495

Model 820
Xerox
1341 West Mockingbird Lane, Dallas, TX 75247
Processor: RS232, Z80
Memory: 64K
Discs: Two 8 inch floppies (400K)
Bus: N/A
Operating System: CP/M
Price: $3,200

ATE

MED

8